BASICMUSIC

PRÁCTICA DE LA ENTONACIÓN Y EL OÍDO MUSICAL

PRACTICING INTONATION AND MUSICAL EAR

TRABAJO DE FIN DE GRADO DEL GRADO EN INGENIERÍA INFORMÁTICA

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

CURSO 2019-2020

AUTORES

ERNESTO PÉREZ MONTALVO

JESÚS VERDÚGUEZ GERVASO

IGNACIO VÍTORES SANCHO

DIRECTOR

MIGUEL GÓMEZ-ZAMALLOA GIL

FACULTAD DE INFORMÁTICA

DEDICATORIA Y AGRADECIMIENTOS

A todos nuestros familiares y amigos que nos han acompañado en estos últimos años y a todos los participantes en este trabajo de fin de grado por su colaboración desinteresada.

RESUMEN

BasicMusic es una aplicación móvil para el sistema operativo Android cuyo objetivo consiste mejorar el oído musical de manera tonal y rítmica y la entonación de la voz. Esto lo consigue de manera divertida e interactiva con una serie de ejercicios propuestos por el equipo de BasicMusic para que el usuario, compitiendo contra sí mismo, vaya mejorando poco a poco y de manera gradual en estos campos.

La interfaz está diseñada para que sea lo más simple e intuitiva posible y así garantizar una gran experiencia al usuario, además de guiarle mediante tutoriales para introducirle en los diferentes modos de juego.

Los ejercicios los podemos dividir en los siguientes modos de juego:

- Adivinar notas: se debe adivinar la nota propuesta.
- Entonación: se debe intentar entonar la nota propuesta.
- Adivinar intervalos: se debe adivinar el intervalo propuesto.
- Crear intervalos: se debe completar el intervalo propuesto dada una nota de inicio.
- Adivinar acordes: se debe adivinar el acorde propuesto.
- Crear acordes: se debe completar el acorde propuesto dada una nota de inicio.
- Dibujar ritmos: se debe indicar el ritmo propuesto en una cuadrícula de botones, que representa la línea temporal.
- Imitar ritmos: se debe imitar en tiempo real el ritmo propuesto pulsando una serie de botones.
- Modo mix: consiste en una mezcla de todos los otros modos a excepción del modo entonación. Se realiza una evaluación al final del modo.

En estos modos de juego, hay niveles que se van desbloqueando y rangos que van aumentando conforme al desempeño del usuario en los ejercicios propuestos. Cuanto más alto sea el nivel, más difíciles serán los ejercicios.

Palabras clave: música, Android, entonación, aplicación, educación, aprendizaje, ritmos, juego.

ABSTRACT

BasicMusic is a mobile application for the Android operating system whose objective is to improve the musical ear in a tonal and rhythmic way as well as the modulation of the voice. This is achieved in a fun and interactive way with a series of exercises proposed by the BasicMusic team so that the user, competing against himself, gradually improves in these fields.

The interface is designed to be as simple and intuitive as possible and thus guarantee a great user experience, in addition to guiding you through tutorials to introduce you to the different game modes.

The exercises can be divided into the following game modes:

- Guess notes: the user must guess a given note.
- Intonation: the user must try to intonate the given note.
- Guess intervals: the user must guess a given interval.
- Create intervals: the user must complete an interval over an initial given note.
- Guess chords: the user must guess a given chord.
- Create chords: the user must complete a chord over an initial note.
- Draw rhythms: the user must click different buttons drawing the given rhythmic pattern.
- Imitate rhythms: the user must imitate the given rhythmic pattern by clicking on different pads.
- Mix: Offers a mix of the different modes, evaluating the user at the end.

In these game modes, there are different levels that unlocks according to the user points and his range that increase according to the user's performance in the proposed exercises. The higher the level, the more difficult the exercises will be.

Key words: music, Android, intonation, application, education, learning, rhythms, game.

ÍNDICE

CAPÍTULO 1 – INTRODUCCIÓN

1.1 – SITUACIÓN ACTUAL Y MOTIVACIÓN

Desde tiempos inmemoriales el arte ha sido una de las piedras angulares de la sociedad, siendo la música una parte importante de este. Hoy en día la música es esencial para incontables personas, ya sea por afición, suponiendo un escape de la rutina, un objetivo secundario en su vida, mera curiosidad o por profesión, dedicando incluso por completo de su vida laboral a este campo, bien de manera vocal, instrumental o realizando trabajos de composición o producción musical.

Cada vez es más frecuente el deseo de aprender esta disciplina, aunque llevarlo a cabo no suele ser fácil debido a falta de medios, falta de motivación o por una razón económica. Por otra parte, educar el oído o la entonación es algo indispensable y necesario para todo aquel que desee tocar algún instrumento, y esto muchas veces lo olvidamos por completo cuando dedicamos demasiado tiempo a la técnica o a la teoría sin tener ningún tipo de criterio. Una de las razones por la que no entrenamos el oído es porque estamos acostumbrados a usar herramientas que nos facilitan el trabajo a la hora de tocar cualquier instrumento. No obstante, grandes profesionales del sector consideran que entrenar el oído puede ayudar a optimizar el aprendizaje de cualquier instrumento, aumentando la versatilidad.

Todos los autores que hemos participado en este trabajo de fin de grado hemos tenido nuestros más y nuestros menos con la música, siempre de manera no profesional, desde aprendizaje autodidacta a estudios en conservatorio. Por lo tanto, sabemos de primera mano los problemas que supone intentar introducirse en el campo y tener la ambición de mejorar en él. Además, al estudiar una carrera basada en la tecnología somos conscientes de las ventajas que esta puede llegar a dar y las mejoras que se pueden conseguir prácticamente en cualquier desempeño. Debido a esto, nuestra principal motivación es facilitar y mejorar el aprendizaje de un arte fundamental de la vida de todas las personas como es la música, por medio de una aplicación móvil de una manera completa, sencilla, accesible y divertida para cualquiera que se lo proponga y tenga esa ambición.

Hemos enfocado nuestros esfuerzos en la mejora de la entonación de la voz y del oído musical tanto a nivel rítmico como a nivel tonal, ya que, según nuestro propio criterio, el de nuestro director y el de nuestros principales entrevistados, estos dos ámbitos son fundamentales para realizar un correcto y completo aprendizaje de la disciplina. Asimismo, son temas que tienen una gran extensión de grados de dificultad, es decir, que ya sea el interesado un principiante o un experto en

la materia, practicar esos dos campos será necesario en su día a día musical. Por último, elegimos estos dos ejercicios porque generalmente se necesita un asistente para confirmar su correcta realización, siendo el asistente una persona conocedora de la materia que pueda supervisar el ejercicio o la ayuda de un instrumento, interpretando con él la secuencia de notas deseada y su posterior comparación, teniendo en cuenta que se necesita conocimiento sobre instrumento.

Por lo que nuestra idea es que la aplicación móvil que desarrollemos tome el papel de ese asistente y ayude a nuestros usuarios a la realización correcta de estas prácticas en solitario, sin contar con personas y/o conocimientos externos.

1.2 – OBJETIVOS

Para entrenar el oído, la entonación y el ritmo musical la mejor forma es practicar, practicar y practicar, por lo que nuestro objetivo principal es proporcionar una herramienta para entrenar dichas habilidades a través de distintas disciplinas mediante una app móvil para Android. La aplicación funcionará como un asistente portátil y ayudará a los usuarios mediante ejercicios y pruebas a mejorar en esos dos ámbitos.

Para ello ofrecemos un amplio abanico de posibilidades organizadas en tres grandes secciones:

- **Adivinar notas musicales**: para entrenar el oído musical trabajando sobre notas individuales, intervalos y acordes en forma de ejercicios de adivinación e intuición.

- **Imitar notas musicales**: para entrenar la entonación en base a pruebas de imitación sobre distintas notas, dependientes del registro vocal del usuario.

- **Ritmos**: para entrenar la percepción de ritmos con la reproducción de patrones aleatorios que el usuario deberá plasmar en una claqueta proporcionada.

1.3 – PLAN DE TRABAJO

El plan de trabajo ha consistido en que los autores hemos tenido una reunión cada una o dos semanas con el director, y hemos avanzado e implementado los requisitos por orden de prioridad.

En las primeras reuniones con el director, se acordaron dichos requisitos analizando cuáles serían los pilares de *BasicMusic*, es decir, qué requisitos no podían falta en el proyecto. Asimismo, se acordó planificar reuniones con diferentes posibles usuarios de la futura aplicación para así consolidar esa lista de requisitos.

En las entrevistas se hicieron las siguientes preguntas:

- ¿Qué es lo básico que esperarías de *BasicMusic*?
- ¿Cuál es la disposición ideal de ejercicios que te gustaría para tu práctica?

- ¿Debería haber ejercicios multijugador?
- ¿Cuál consideras que debería ser el nivel de dificultad máximo?
- ¿Debería de haber exámenes/pruebas para subir el nivel del usuario?
- ¿Deberíamos mostrar datos numéricos al usuario (porcentaje de entonación, tiempo tardado...)?
- ¿Debería ser posible la escucha de diferentes instrumentos?

Posteriormente, desarrollamos la siguiente lista de requisitos:

- Oído musical
    - Función para identificar una nota
    - Función para identificar un intervalo
    - Función para identificar un acorde
    - Función para realizar intervalos
    - Función para realizar acordes
- Entonación de la voz
    - Función para entonar notas musicales en base a tu rango vocal
- Ritmos
    - Función para identificar ritmos aleatorios
    - Función para realizar ritmos aleatorios

En la siguiente reunión con el director pusimos en común la lista de requisitos. Una vez adquirida su confirmación nos comenzamos a hacer la siguiente pregunta: ¿cuál es el mejor lenguaje y *framework* para una correcta implementación de la aplicación?

Con la decisión tomada, empezamos a realizar un estudio de aplicaciones similares para adquirir ideas para un correcto diseño de la interfaz de la aplicación y analizar sus funcionalidades.

Una vez tuvimos claros los requisitos y cómo implementarlos, nos pusimos manos a la obra. Debido a que ningún miembro de nuestro grupo tenía conocimientos de programación en el entorno de Android el comienzo fue difícil, pero una vez que investigamos y practicamos en pequeños proyectos con funcionalidades concretas comenzamos con la implementación de los requisitos de “Función para identificar una nota” y “Función para entonar notas musicales”.

Cuando todos nos introducimos en el entorno, nos dividimos los requisitos para así agilizar el proceso de implementación. Uno de nosotros se encargó de los requisitos “Función para identificar un intervalo” y “Función para identificar un acorde”, otro se encargó de los requisitos “Función para realizar intervalos” y “Función para realizar acordes” y por último otro de los miembros se encargó

de los requisitos “Función para identificar ritmos aleatorios” y “Función para realizar ritmos aleatorios”.

Realizada la implementación de todos los requisitos, revisado su correcto funcionamiento y comprobada sus funcionalidades con el director, nuestro principal objetivo fue adaptar la dificultad para el más amplio abanico de usuarios posible. Para ello implementamos una serie de tutoriales y facilidades para simplificar el uso de la aplicación.

Por último, implementamos lo que nosotros llamamos “Modo Mix”, el cual consiste en un conjunto aleatorio de ejercicios de los requisitos anteriormente mencionados, todos en una misma actividad, exceptuando los ejercicios del requisito “Función para entonar notas musicales en base a tu rango vocal” debido a la gran diferencia de jugabilidad.

1.4 – ESTRUCTURA DEL DOCUMENTO

La estructura del documento está dividida en 10 capítulos:

- Capítulo 1. Introducción: explicación de la motivación para realizar este proyecto, cuáles fueron los objetivos que nos marcamos, y cuál fue el plan que seguimos para hacerlos.

- Capítulo 2. Introduction: traducción del capítulo anterior al inglés.

- Capítulo 3. Origen de la aplicación: en este capítulo explicamos los orígenes de la aplicación; cómo surgió la idea, cuáles fueron las aplicaciones que sirvieron de inspiración y de apoyo para su desarrollo y las entrevistas realizadas para definir la base y los requisitos del proyecto.

- Capítulo 4. Investigación: describe la investigación previa a la implementación del proyecto, tanto de teoría musical como qué entorno de desarrollo iba a ser el óptimo para una aplicación de estas características.

- Capítulo 5. Implementación: se realiza una explicación más en profundidad de la implementación de la aplicación, incluyendo la base de datos que empleamos, la arquitectura de la aplicación, los permisos que solicitamos para su correcto funcionamiento, un informe detallado de la interfaz de cada pantalla importante dentro de la aplicación, los paquetes que usamos y la implementación del código de estos.

- Capítulo 6. Funcionalidad: en este capítulo se expone un manual de usuario en profundidad del proyecto, en el que se explican todos los modos de juego y su funcionalidad, además de una exposición del sistema de dificultad y puntuación interna.

- Capítulo 7. Aportación individual: se presenta una descripción de la aportación individual de cada miembro del grupo al proyecto.

- Capítulo 8. Conclusión: se indica la ubicación del repositorio del proyecto y dónde puede ser instalado, así como los objetivos conseguidos, posibles mejoras futuras y satisfacción personal.

- Capítulo 9. In conclusion: traducción del capítulo anterior al inglés.

- Capítulo 10. Bibliografía: incluye las fuentes y código externo utilizado en el proyecto.

CHAPTER 2 – INTRODUCTION

2.1 – CURRENT SITUATION AND MOTIVATION

Since time immemorial, art has been one of the cornerstones of society, with music being an important part of it. Nowadays music is essential for countless people, being a hobby, assuming an escape from routine, a secondary objective in their life, mere curiosity, or being a profession, dedicating their working life to this field, vocally, instrumentally or making musical production work.

The desire to learn this discipline is more and more frequent, although carrying it out is not easy due to the lack of means, lack of motivation, or for an economic reason. On the other hand, educating the ear and voice modulation is something indispensable and necessary for anyone who wants to play an instrument, and this is often forgotten when we spend too much time on technique or theory without having any kind of criterion. One of the reasons why we do not train the ear is because we are used to using a large number of tools that make it easier for us to play any instrument, however, great professionals in the sector consider that training the ear can help optimize the learning of any instrument, increasing versatility.

All the authors who have participated in this project have had our things with music, in a non-professional way, from self-taught learning to conservatory studies. Therefore, we know firsthand the problems of trying to enter the field and have the ambition to improve on it. In addition, since we are studying a career based on technology, we are aware of the advantages that it can give and the improvements that can be achieved in practically any situation. Based on that, our main motivation is to facilitate and improve the learning of a fundamental art of life for all people, such as music, through a mobile application, in a complete, simple, accessible and fun way, for anyone having that ambition.

We have focused our efforts on improving the modulation of the voice and the musical ear, both at the rhythmic level and at the tonal level, since, according to our own criterion, that of our supervisor and that of our main interviewees, these two areas are fundamental to carry out a correct and complete learning of the discipline. Likewise, these are subjects that have a great extension of degrees of difficulty, whether the interested one is a beginner or an expert in the subject. We choose these two exercises because an assistant is generally needed to confirm their correct performance, the assistant being a person who is knowledgeable about the subject who can supervise the exercise, or the help of an instrument playing with him the desired sequence of notes and their subsequent comparison, bearing in mind that knowledge about the instrument is required.

So, our idea is that the mobile application that we develop, take the role of that assistant and help our users to carry out these practices alone, without having to count on external people.

2.2 – OBJECTIVES

The best way to train musical ear and intonation is to practice a lot, so our great objective is to provide a tool to train these skills through different disciplines. We will do it using a mobile app for Android. The application will work as a portable assistant and will help users through exercises and tests to improve in those two areas.

For that we offer a wide range of possibilities organized in three large sections:

- **Guess the note:** In order to train the musical ear, working on individual notes, intervals and chords in the form of guessing and intuition exercises.

- **Intonation**: In order to train intonation based on imitation tests on different notes, depending on the user's vocal register.

- **Rhythm**: In order to train the perception of rhythms with the reproduction of random patterns that the user must capture in a proportioned clapperboard.

2.3 – WORKPLAN

The authors have had a meeting every one or two weeks with the director, and we have progressed and implemented his requirements in order of priority.

In the first meetings, we agreed on those requirements, analyzing what would be the pillars of BasicMusic and what requirements could not be missing in our assistant. Likewise, it was agreed to plan meetings with possible users of the application in order to consolidate that list of requirements.

The following questions were asked in the interviews with those users:

- What are the basics you would expect from *BasicMusic*?
- What is the ideal set of exercises that you would like for practice?
- Should there be multiplayer exercises?
- What do you think is the maximum difficulty level?
- Should there be exams to test the user?
- Should the app show the user’s statistics?
- Is it important to allow the reproduction of different sets of instruments?

Subsequently, we developed the following list of requirements:

- Musical ear:
    - Function to guess a note.
    - Function to guess an Interval.
    - Function to guess a chord.
    - Function to create intervals.
    - Function to create chords.
- Voice intonation
    - Function to match musical notes based on the user's vocal register.
- Rhythm
    - Function to guess random rhythmic patterns.
    - Function to create rhythmic patterns.

At the next meeting with the supervisor we shared our first list of requirements, and once we had confirmed it with him, we began to wonder the following question: What is the best programming language and framework for the implementation of our application?

Once we made a decision, we began to look for similar apps to obtain ideas of a nice design for our app interface, and to analyze its functionalities.

After we were clear about the requirements and how to implement them, we got down to work. Since no member of our group had programming knowledge in the Android environment, the beginning was difficult, but once we researched small projects with specific functionalities, we implemented the requirements of "Function to guess a note" and "Function to intonate musical notes".

When we all knew about the environment, we divided the requirements between us to streamline the implementation process. One of us took care of the requirement "Function to guess an interval" and "Function to guess a chord", another one took care of the requirement "Function to create an interval" and "Function to create chord" and finally the last member took care of the requirement "Function to guess random rhythmic patterns." and " Function to create rhythmic patterns ".

Once all the requirements were nicely and correctly implemented, and their functionalities were verified with the director, our main objective was to adapt the difficulty to the widest range of users as possible. In order to do this, we implemented a series of tutorials and facilities to simplify the use of the application.

Finally, we implemented what we call "Mix Mode" which consists of a random set of exercises.

2.4– DOCUMENT'S STRUCTURE

The structure of the document is divided into 10 chapters:

- Chapter 1. Introduction: Explanation of the motivation to carry out this project, the objectives that we set ourselves and the plan that we followed to do them.

- Chapter 2. Introduction: English translation of the previous chapter.

- Chapter 3. Origins of the application: In this chapter we explain the origins of the application, how we came up with the idea, what were the applications that inspired us and supported us in its development, and the interviews we conducted to define the basis and requirements of the project.

- Chapter 4. Research: Explanation of all the previous research we carried out before starting the implementation of the project, from music theory, to the development environments we looked to find the optimal one for an application of these characteristics.

- Chapter 5. Implementation: We carry out a more in-depth explanation of the implementation of the application, including the database that we use, the architecture of the application, the permissions that we request for its correct operation, a detailed report of the interface of each screen, the packages we use and their implementation.

- Chapter 6. Functionality: In this chapter we carry out an in-depth user manual of the project, in which we explain all the game modes and their functionality, as well as an exposition of the difficulty and scoring system.

- Chapter 7. Individual contribution: Description of the individual contribution of each member to the project.

- Chapter 8. Conclusion: We show the location of the project repository and indicate where it can be installed, as well as the objectives achieved, possible future improvements and personal satisfaction.

- Chapter 9. Conclusion: English's translation of the previous chapter.

- Chapter 10. Bibliography: Sources and external code used in the project

CAPÍTULO 3 – ORIGEN DE LA APLICACIÓN

3.1 – INTRODUCCIÓN

Como hemos mencionado antes, todos los miembros del grupo hemos estado en contacto de alguna manera con la música. Por lo tanto, nos hicimos la siguiente pregunta: ¿cómo podemos ayudar a gente como nosotros a introducirse en la música, con un amplio rango de dificultades para que sirva tanto a principiantes como a expertos, y con ámbito útil dentro de esta?

La primera respuesta que se nos ocurrió fue el oído musical, ya que es una aptitud fundamental que está presente en la música a todos los niveles.

Posteriormente nos vino a la cabeza la entonación de la voz, debido a que es una de las habilidades más demandadas y más difíciles de practicar para una persona en solitario sin ningún tipo de conocimiento musical.

Y, por último, durante el desarrollo de las dos anteriores, caímos en la cuenta de que el ritmo es algo básico en la música y podríamos extender la funcionalidad para que abarcase también ese campo.

3.2 – ESTUDIO DE MERCADO

Previo al desarrollo de la aplicación realizamos un estudio de mercado para orientarnos en cuanto los requisitos y el diseño de la interfaz para una aplicación de estas características, así como para encontrar la forma de mejorar las opciones disponibles.

Cabe destacar que en este apartado distinguimos dos tipos de usuarios: el usuario novel (usuario que entra por primera vez a la aplicación en cuestión) y el usuario experimentado (usuario que ha usado la aplicación durante mínimo unas horas, y que sabe cómo funciona la aplicación en general).

3.2.1 – *FL Studio* y *BandLab* [1][2]

FL Studio y *BandLab* son DAW que hemos usado a nivel de usuario, y consideramos que tiene una interfaz intuitiva para el usuario experimentado además de muy atractivas y óptimas, aunque cabe destacar que un usuario novel, debido a la falta de práctica y conocimientos se siente un poco perdido debido a la ausencia de tutoriales eficaces y amigables.

3.2.2 – *Smartick* [3]

Smartick es una aplicación de aprendizaje de matemáticas para niños de primaria que comparte con nuestra aplicación el objetivo de enseñar en un área de conocimiento determinado, a través de pruebas individuales que examinan al usuario para que éste demuestre lo que ha aprendido.

Presenta un sistema de progresión por fallos y aciertos similar a nuestra aplicación, en este caso se le presenta al usuario una serie de niveles que éste debe superar para continuar avanzando.

Finalmente, *Smartick* cuenta con un sistema de usuarios ideado para que cada miembro de la familia o varios amigos puedan crearse un usuario distinto y progresar de forma individual, cambiando de usuario con facilidad, característica que comparte con nuestra aplicación

3.2.3 – *Music Tutor* [4]

El objetivo de *Music Tutor* es instruir al usuario en el solfeo y la lectura de notas musicales, para ello le presenta un modo de juego donde la aplicación muestra una nota en un pentagrama con Clave de Sol, Fa o Do, y el usuario debe acertarla de entre una serie de opciones posibles, dentro de un límite de tiempo.

La aplicación también proporciona una pantalla de estadísticas que nos resultó interesante desde el primer momento, ya que es una forma intuitiva de poder observar el progreso del usuario y su desempeño en las pruebas, por lo que decidimos incluir una pantalla similar en nuestra aplicación.

Por otra parte *Music Tutor* es muy simple en su cometido, ya que cuenta con un solo modo de juego y no presenta sistemas de progresión ni de múltiples usuarios, mientras que la idea de nuestra aplicación es proporcionar al usuario varios modos de juego de forma que este pueda elegir el adecuado según la disciplina que quiera practicar, desde oído musical hasta entonación, así como darle un seguimiento personal de su progreso, motivándole en la medida de lo posible mediante un sistema de progresión basado en rangos.

3.2.4 – *Music Trainer* [5]

Es aplicación Android similar a *Music Tutor* cuyo objetivo es la práctica de la lectura de notas musicales. También proporciona al usuario una nota y éste debe adivinar su nombre de entre una serie de opciones posibles, la aplicación cuenta además con un contador de notas adivinadas y un sistema de récord sobre dicho contador.

Aunque es más simple que *Music Tutor*, *Music Trainer* comparte aspiración con nuestra aplicación, no obstante, no proporciona tantos modos de juego y está estrictamente limitada a la lectura musical.

3.2.5 – *Complete Rhythm Trainer* [6]

Es una aplicación enfocada al entrenamiento de ritmos que proporciona al usuario una serie de modos de juegos relacionados con la imitación, creación o adivinación de ritmos.

Presenta un amplio abanico de opciones de ritmos (Compases, figuras etc.…) distintas para cada modo organizadas en distintos niveles, con un sistema de dificultad similar al de nuestra aplicación.

No obstante, no tiene sistema de progresión por rangos como nuestra aplicación y en su lugar implementa un sistema de Logros que el usuario irá completando a medida que avanza en los niveles y que le darán una serie de puntos.

Por otra parte, la aplicación está exclusivamente limitada a la práctica de ritmos, no ofreciendo modos de oído musical o entonación como ofrece nuestra aplicación.

3.2.6 – *Complete Ear Trainer* [7]

Es una aplicación similar a la anterior, pero en este caso enfocada al entrenamiento del oído musical. Proporciona al usuario tres modos de juego: Modo Fácil, Clásico y Arcade, cada uno ofreciendo una serie de capítulos desbloqueados según se aumenta la puntuación en el capítulo actual.

Según el modo de juego los capítulos estarán formados por pruebas de distinta índole, siendo menos variadas en el Modo Fácil y más variadas en el modo Clásico, con la presencia de acordes y arpegios.

El sistema de puntuación y desbloqueo de niveles es similar al de nuestra aplicación, pero, al igual que *Complete Rhythm Trainer*, *Complete Ear Trainer* no ofrece un sistema de rangos y en su lugar presenta un sistema de logros, en nuestra opinión menos intuitivo.

3.2.7 – *Music Trainer Professional* [8]

Es una aplicación basada en la práctica del oído y lectura musical, presenta una serie de modos que van desde práctica de notas musicales hasta práctica de armaduras o intervalos; no obstante, solo el primer modo mencionado forma parte de la versión gratuita de la aplicación. Dicho modo presenta una nota sobre un pentagrama en Clave de Fa o Sol, y el usuario debe pulsar sobre un piano en pantalla la tecla correspondiente a la nota en cuestión.

La presencia de un piano digital en el modo de práctica de notas musicales supone una ventaja para aquellos usuarios que quieran practicar piano, ya que ayuda a localizar las distintas notas musicales en el instrumento.

Al igual que el resto de las aplicaciones mencionadas, *Music Trainer Professional* está limitada en cuanto a sus modos de juego, ya que no presenta tantos como nuestra aplicación ni tiene un sistema de progresión, niveles o puntuaciones e implementa compras digitales.

3.3 – ENTREVISTAS

Al comienzo del desarrollo realizamos una serie de entrevistas a tres posibles usuarios de nuestra futura aplicación. En cada una de estas entrevistas, con previo consentimiento de los entrevistados, se tomaron notas de las respuestas para posteriormente consultarlas con el objetivo de optimizar el desarrollo.

3.3.1 – Primera entrevista

La primera entrevista que realizamos fue a Carmen Móxo y Alberto Jódar.

Carmen se dedica a la música de una manera no profesional, aunque ha participado en varios discos y producciones musicales además de haberse formado en el conservatorio y escuelas musicales.

Alberto tiene un perfil parecido al de Carmen utilizando internet tanto como medio de aprendizaje como de difusión de su pasión por la música.

Los perfiles de Carmen y Alberto los consideramos de gran interés ya que tienen un conocimiento musical medio y les gustaría aprender y mejorar de una manera individual, por lo tanto, pueden llegar a ser usuarios potenciales de nuestra aplicación.

Debido a que fue la primera entrevista que realizamos, con ellos intentamos asentar todas las bases y límites de nuestra aplicación para así poder tener una idea más clara y realista del proyecto. Les consultamos qué esperarían de una aplicación como la nuestra, cuál consideran un método de evaluación efectivo y justo, además de otros muchos detalles que para ellos deberían estar obligatoriamente en la aplicación.

3.3.3 – Segunda entrevista

La segunda entrevista se la realizamos a Montserrat Egea [9].

Montserrat es chelista profesional que ha actuado en diferentes orquestas famosas como la Orquesta Sinfónica de RTVE [10] o la orquesta sinfónica de Musikene [11]. Actualmente se está formando profesionalmente en la Escuela Superior de Música Reina Sofía [12].

El objetivo de la entrevista con Montserrat fue ajustar diversos ámbitos más específicos y complejos musicalmente hablando debido a su gran conocimiento en el campo. Además, sirvió para orientarnos sobre qué temas musicales deberíamos tratar en la aplicación, en qué consistían y cómo podrían ayudar en el aprendizaje. Nos dio diversos consejos sobre teoría musical que nos ayudaron posteriormente a la hora del desarrollo.

No consideramos a Montserrat como una usuaria potencial de nuestra aplicación, debido a que, con su formación, nuestra aplicación le podría resultar trivial. Sin embargo, consideramos que una visión bastante más profesional y superior de nuestro usuario objetivo podría ayudarnos para establecer una idea correcta de formación y evaluación de usuarios que buscan usar nuestra aplicación para tener el conocimiento musical más amplio.

CAPÍTULO 4 – INVESTIGACIÓN

4.1 – TEORÍA MUSICAL

Nuestro primer objetivo antes de comenzar a implementar la aplicación fue aprender y conseguir conocimientos sobre teoría musical, ya que nuestra aplicación necesita de dichos conocimientos para desarrollar los niveles, y nosotros, como desarrolladores, tenemos que saber cómo enfocar el aprendizaje de cara al entrenamiento del oído y la entonación usando los distintos conceptos de teoría musical.

Para conseguir los conocimientos contamos con la ayuda de unas diapositivas que nos proporcionó nuestro director [13] y que, junto con búsquedas en la web [14], nos permitieron conocer los conceptos y detalles de los pilares de la teoría musical para nuestra aplicación.

Dichos pilares son:

- Las diferentes notas musicales, con sus frecuencias correspondientes y las distintas octavas.
- Los intervalos musicales, su definición y los distintos tipos.
- Los acordes, su definición y los distintos tipos.
- Los distintos compases y las diferencias entre ellos, así como el tiempo musical.

Una vez que consideramos que teníamos una cierta idea sobre los conceptos clave, comenzamos a investigar los métodos de implementación de nuestra aplicación que más se adaptasen a los que buscábamos, lo que no fue tarea fácil.

Hay que destacar que nuestro aprendizaje de teoría musical no se quedó solo en lo que hicimos antes de comenzar a implementar, sino que ha continuado desde entonces, profundizando en los aspectos que más usamos en nuestra aplicación y recibiendo formación de las clases de la asignatura optativa de informática musical.

4.2 – PYTHON - KIVY

Una vez tuvimos los conocimientos de teoría musical necesarios, comenzamos a indagar en las distintas posibilidades para desarrollar nuestra aplicación, pasando por distintos lenguajes de programación y entornos de desarrollo, probando y analizando cada uno de ellos para encontrar la mejor opción.

Comenzamos investigando con *Python*, lenguaje que los tres controlábamos y donde podíamos usar las bibliotecas *PyAudio* [15] y *Numpy* [16] para el análisis y la síntesis de audio, una parte muy importante de nuestra aplicación.

No obstante, nos acabamos enfrentando a la falta de entornos y bibliotecas que garantizaran un desarrollo de aplicaciones Android sostenible. Con el tiempo y tras buscar mucho, encontramos una biblioteca que nos podía ser útil. Se trataba de *Kivy* [17], una biblioteca de *Python* gratuita y de código abierto para desarrollar aplicaciones móviles y otro software de aplicaciones multitáctiles con una interfaz de usuario natural.

Intentamos programar un grabador/reproductor que funcionase en un dispositivo Android, generando el *apk* a través de una herramienta para Linux llamada *Buildozer* [18], que permite generar *apks* y controlar sus versiones de una forma sencilla; no obstante, tuvimos bastantes problemas de compatibilidad para usarlo con la biblioteca *Kivy* y, aunque conseguimos desarrollar un grabador/reproductor funcional en el ordenador, no conseguimos hacerlo funcionar en un dispositivo Android. Tras varios intentos, decidimos abandonar *Python* y *Kivy* para continuar buscando otra alternativa.

4.3 – JUCE - C++

La siguiente posibilidad que nos planteamos a propuesta de nuestro director fue *JUCE* [19], un marco de aplicación de código abierto escrito en *C++* útil para el desarrollo de aplicaciones de audio. Consideramos esta posibilidad por su enfoque en elementos de audio como síntesis o análisis, que era justo lo que necesitábamos.

Tras investigar la mejor forma de desarrollar aplicaciones móviles con JUCE llegamos a la conclusión de que la manera óptima era integrar JUCE y Android Studio debido a la facilidad de este último para el desarrollo de interfaces gráficas, gracias a la posibilidad de observar los cambios en la interfaz en tiempo real con un sistema “Drag & Place” y a la facilidad de generación del apk para el testeo inmediato.

El método escogido nos obligó a buscar formas para integrar y ejecutar código C++ en Android Studio [20], un IDE pensado únicamente para trabajar con *Java* y *Kotlin*. Esta búsqueda nos llevó a considerar JNI (*Java Native Interface*), un mecanismo que permite que métodos nativos de C++, implementados por separado en ficheros .cpp, interactúen con el programa Java mediante una biblioteca de enlace dinámico. En otras palabras, *JNI* es una suerte de puente de comunicación entre un programa Java y código C++ (*JUCE*).

No obstante, con el tiempo comprobamos que la sincronización entre *JUCE* y *Android* Studio no funcionaba correctamente en *Windows*, por lo que, para solucionarlo, los tres nos vimos obligados a usar particiones *Linux* para ejecutar correctamente *Android Studio* con *JUCE*.

En primer lugar, intentamos crear un grabador/reproductor similar al del apartado anterior que fuese funcional en un dispositivo móvil, pero finalmente tampoco tuvimos éxito, debido en parte, a incompatibilidades del emulador de *Android Studio* con algunas de las bibliotecas de *JUCE*, que, sumado a la extrema inestabilidad del montaje en nuestras distribuciones de *Linux* acabó por obligarnos a buscar otro acercamiento.

4.4 – JAVA - JFUGUE

A continuación, tras los diversos problemas que tuvimos con otros lenguajes bien para desarrollar aplicaciones de audio en dispositivo móviles o bien para integrarlos con Android Studio, decidimos comenzar a investigar en *Java*, lenguaje que sabíamos que Android Studio soportaba sin problema.

En primer lugar, nos dedicamos a buscar bibliotecas que pudieran sernos útiles para nuestra aplicación, enfocándonos sobre todo en aquellas que tuvieran funciones y algoritmos para el análisis y la síntesis de audio.

Una de las bibliotecas más destacadas fue *JFugue* [21], una librería de código abierto que permite analizar y sintetizar audio de forma sencilla en *Java*. *JFugue* nos proporcionaba una manera intuitiva de generar audio y ritmos dinámicamente, así como analizar parámetros de audio entrante, por lo que consideramos que era perfecta para nuestra aplicación.

No obstante, aparecieron problemas a la hora de integrarlo con Android y, tras investigarlo, descubrimos que *JFugue* trabajaba sobre unas librerías de *Java* (*Javax.Sound*) no soportadas en la última versión en *Android*, por lo que no podíamos usar la biblioteca en *Android Studio*. Conscientes de las facilidades que nos podía aportar *JFugue* continuamos buscando una forma de usarlo en *Android Studio*, y descubrimos en GitHub una adaptación de *JFugue* para Android que prescindía de las últimas versiones de *Javax.Sound*. Sin embargo, para hacerlo funcionar había que modificar el *Gradle* y tras mucho intentarlo, no fuimos capaces de estabilizar el funcionamiento de *JFugue* para Android debido a una gran cantidad de incompatibilidades con la última versión de *Android Studio*, por lo que decidimos abandonar *JFugue*.

4.5 – ANDROID STUDIO

Tras el fracaso de *JFugue*, continuamos buscando una herramienta Java que nos permitiese programar síntesis y análisis de audio de forma efectiva en *Android Studio*.

Finalmente encontramos *TarsosDSP* [22], una biblioteca *Java* diseñada para el procesamiento de audio en tiempo real que además no tenía ninguna dependencia de *Javax.Sound*, por lo que su uso en *Android* era completamente viable. Tras programar una aplicación analizadora de frecuencia en *Android* usando *TarsosDSP* y asegurarnos de su correcto funcionamiento, decidimos comenzar a desarrollar la aplicación sobre dicha biblioteca en *Android Studio*.

CAPÍTULO 5 – IMPLEMENTACIÓN

5.1 – INTRODUCCIÓN

En este capítulo se detalla la implementación de la aplicación. Se describirá la arquitectura empleada y la base de datos, los aspectos relacionados con la interfaz de usuario, la estructura del código y su relación con las distintas funcionalidades de *BasicMusic*.

5.2 – ARQUITECTURA DE LA APLICACIÓN

La aplicación se compone de una base de datos autocontenida y el propio cliente, que es el encargado de realizar las consultas a la base de datos y llevar el flujo de actividades.

Para la base de datos hemos usado la capa de abstracción *Room [23],* que emplea internamente *SQLite,* proporcionando una capa de abstracción para realizar todas las operaciones en el cliente.

La implementación del cliente se ha realizado íntegramente en *Android Studio*, empleando como lenguaje de programación *Java* y archivos *XML* para los aspectos gráficos. Por tanto, la aplicación es compatible con los dispositivos con sistema operativo *Android*.

5.3 – IMPLEMENTACIÓN DE LA BASE DE DATOS

Tras definir la información que queríamos mostrar al usuario sobre cada modalidad de uso, decidimos que la base de datos debía estar compuesta por cuatro entidades. Al ser todas las relaciones del tipo 1 a N, no se necesitan tablas auxiliares para implementarlas.

- Tabla usuario: almacena la información relativa a la cuenta del usuario y si su sesión debe ser recordada. Como clave primaria hemos empleado el correo electrónico.

- Puntuación: almacena la puntuación, el rango y el nivel de un usuario en un modo de juego específico. Guarda también si un usuario ha iniciado el modo o es su primera vez.

- NivelImitar: almacena la información referente a los niveles del modo Entonación. Debido a que la información a almacenar de estos niveles es distinta a todos los demás, decidimos que era preferible realizar una tabla exclusiva en lugar de sobrecargar una única tabla que almacenase todos.

- NivelAdivinar: almacena la información del resto de modos de juego de la aplicación.

Para la gestión de la base de datos hacemos uso de tres clases DAO (Data Access Object) diferentes. Cada una contiene las operaciones CRUD (Create, Read, Update, Delete) básicas.

- DAONivel: se encarga de la gestión de las tablas NivelAdivinar y NivelImitar.
- DAOUsuario: se encarga de la gestión de la tabla Usuario.
- DAOPuntuacion: se encarga de la gestión de la tabla Puntuación.

5.4 – PERMISOS DE LA APLICACIÓN

Debido a la funcionalidad de la aplicación, es necesario acceder al micrófono del dispositivo para poder recoger muestras de audio. Por esta razón, al iniciar la aplicación se solicita el permiso al usuario si no lo ha otorgado previamente. Este requisito puede verse en el archivo AndroidManifest.xml.5.6 – Estructura del proyecto

5.6.1 – Paquete *BBDD*

En este paquete incluimos todas las clases relacionadas con el modelo de la aplicación y su gestión a excepción de la clase *GestorBBDD*. La transformación de clases Java a entidades propias de la base de datos se realiza mediante anotaciones *Room*, que es una capa de abstracción para facilitar la implementación del modelo similar a JPA. Estas clases son:

- Las entidades (definidas en el capítulo 5.3) se encuentran representadas por sus clases Java que incluyen la anotación @Entity. Las clases contienen las operaciones básicas de acceso a los atributos y una constructora con los parámetros.
- También se incluyen interfaces marcadas con la anotación @Dao para definir los DAOs y las operaciones CRUD, marcadas con las anotaciones @Insert, @Update, @Delete y @Query (operaciones de consulta).
- Dentro del paquete existe otra clase abstracta denominada AppDatabase, que es la encargada de crear la base de datos al iniciar la aplicación por primera vez y proporciona un punto de acceso a ella desde otras partes de la aplicación. En ella se determinan las entidades que forman la base de datos. Contiene atributos de tipo DAO y una instancia de la propia clase que permite acceder a ellos.

5.6.2 – Paquete *Enumerados*

El paquete *enumerado* está compuesto por todos los enumerados que utilizamos en el proyecto. Estos enumerados son:

- *Acordes:* Contiene todos los acordes que aparecen en la aplicación, representados por sus intervalos entre notas. Introducimos el método *devuelveNotasAcorde* que devuelve en un array las notas del acorde que nosotros le indiquemos dada una nota de inicio y una octava.

- *Dificultad*: Contiene las dificultades que planteamos en los diferentes niveles.

- *DuracionSonido:* Contiene la representación de la duración de las notas musicales, mediante silencios. Se emplea en el modo ritmos.

- *Intervalos:* Contiene todos los intervalos que aparecen en nuestro proyecto. El enumerado está formado por el nombre del intervalo y la distancia en semitonos.

- *ModoJuego:* Este enumerado contiene todos los modos de juego de *BasicMusic*. Introducimos los métodos para mostrar y rellenar el texto del pop up que se muestra al cambiar el nivel.

- *NivelMix:* Este enumerado está formado por los niveles del modo mix. Cada nivel contiene sus parámetros.

- *Notas:* Contiene los 12 semitonos musicales que existen, sus frecuencias admitidas en el modo entonación, su frecuencia exacta y el path al fichero que lo contiene.

- *Octavas:* Contiene las octavas, desde la primera hasta la séptima.

- *PuntosNiveles:* En este enumerado se encuentran todos los niveles que utilizamos, además del mínimo y máximo de los puntos necesarios para subir o bajar de nivel. Incluimos el método *devuelveNivel* que devuelve el nivel dada una puntuación y un modo de juego.

- *RangosPuntuaciones:* Aquí podemos encontrar todos los rangos de BasicMusic, junto con sus imágenes. Sus métodos más importantes son *actualizaRango,* que actualiza el rango del usuario en base su puntuación y el método *mostrar_popUp_rango,* que muestra un pop up al subir de rango.

- *RangosVocales:* hemos optado por introducir rangos vocales como *Niño, Mujer y Hombre* para facilitar el correcto uso del modo de juego *Entonación,* pero en realidad nos basamos en los rangos vocales musicales como Tenor, Soprano, etc. Contiene las octavas y los semitonos de inicio y final en cada rango.

5.6.3 – Paquete *Singletons*

Debido a que teníamos mucha funcionalidad común en los diferentes modos de juego, vimos muy recomendable aplicar el patrón Singleton para centralizar la implementación de la parte lógica de la distribución de niveles, la gestión de los reproductores, el control de las estadísticas del usuario y el desarrollo del Modo Mix. Las clases contenidas en este paquete son las siguientes:

- *Controlador*: esta clase es la encargada de almacenar el nivel/dificultad y modo de juego seleccionados por el usuario para establecer los parámetros que determinan los cambios en

la prueba a realizar (conjunto del que sacar las opciones, número de opciones, longitud y tempo de los ritmos…).

El método más importante de esta clase es el encargado de establecer la dificultad. Se distinguen dos formas diferentes de realizar esto:

1. **Modos** dentro de la categoría **adivinar audio (cap. 6.1.1)**: una vez que el usuario elige un nivel dentro de un determinado modo, se establece el subconjunto de notas, octavas y acordes (en función del modo) sobre el que se escogerán las respuestas. Estos subconjuntos se incrementan de manera progresiva hasta alcanzar el conjunto de todas las posibilidades en el nivel máximo.

2. **Modos** dentro de la categoría **ritmos (cap. 6.1.3):** en estos modos, los parámetros que varían en función del nivel son la longitud del ritmo, el número de instrumentos que tiene el ritmo y la pausa entre cada fragmento del compás.

   En los primeros niveles, la longitud de los ritmos es más pequeña. La pausa, sin embargo, no sigue una progresión lineal debido a que cada dos niveles incluimos un nuevo instrumento en el ritmo. Por ello, creímos que era conveniente ralentizar el ritmo en el primer nivel en el que se introduce ese nuevo instrumento (niveles impares) y acelerarlo en el segundo.

- *ControladorMix:* esta clase es la encargada de llevar el desarrollo del *Modo Mix.* En ella se almacenan las pruebas que debe realizar el usuario ordenadas según su aparición, el nivel del modo de juego, el resultado de cada prueba y el índice de la prueba que está realizando el usuario.

  A mayor nivel seleccionado por el usuario aumenta tanto la dificultad individual de las pruebas, como el porcentaje total de pruebas acertadas que se requieren para superar el *Mix*. Para determinar la dificultad individual de cada prueba que compone el *Mix*, se apoya en el uso de la clase *Controlador*, estableciendo el nivel según corresponda para cada modo de juego (establecido en el enumerado *NivelMix*).

- *GestorBBDD:* esta clase es la encargada de controlar toda la lógica de la aplicación relativa a la gestión de la base de datos. Para realizar las operaciones contiene un parámetro de la clase *AppDatabase.* Las funciones principales que lleva a cabo son las siguientes:

  1. Inserción o actualización de los datos anteriores relativos a un nivel según corresponda. También modifica la puntuación del usuario en ese modo de juego.
  2. Validación de la contraseña introducida, mantener activa la sesión del usuario si éste lo ha indicado al realizar el login y cerrar la sesión del usuario.
  3. Registrar un nuevo usuario en la base de datos, comprobando que no existía previamente ningún otro usuario con el mismo correo.
  4. Dar formato y devolver las estadísticas de cada modo de juego para poder mostrar al usuario sus datos en forma de tabla.

5. Comprobar si es la primera vez que un usuario realiza un modo específico de juego para mostrar los pop-ups de ayuda. Se realiza el mismo procedimiento la primera vez que se inicia la aplicación.

- *FactoriaNotas:* esta clase se encarga de la generación aleatoria de notas, intervalos y acordes. También almacena las rutas de los ficheros que contienen las notas de referencia que se ofrecen como ayuda en algunos niveles. Las funciones más importantes que contiene son las siguientes:
    1. Función *getNumNotasAleatorias*: función que devuelve un conjunto de notas de un tamaño prefijado.
    2. Función *getNotasIntervalo*: función devuelve el par de notas que forman un intervalo.
    3. Función *getNotasRV:* dado un rango vocal y un número de notas, esta función devuelve un conjunto de notas pertenecientes a ese rango.
    4. Función *devuelveAcordes:* función que devuelve todos los acordes posibles formados a partir de una nota específica.

5.6.4 – Paquete *Acordes*

El paquete Acordes agrupa toda la lógica relacionada con el modo Acordes de la aplicación, tanto el modo Adivinar Acorde como el modo Crear Acorde.

El paquete está formado por las siguientes clases y subpaquetes:

- Clase *SeleccionarModoAcordes*: sirve de puente entre los dos modos de juego y establece uno u otro, además de mostrar la imagen del rango que tiene el usuario.

- Clase *TutorialAdivinarAcordes*: accesible a través del botón Info disponible en los primeros niveles, sirve para mostrar una ayuda con todos los acordes posibles.

A continuación, encontramos dos subpaquetes, uno para cada modo de juego: *Crear* y *Adivinar*:

El subpaquete *Adivinar* contiene las siguientes clases:

- Clase *AdivinarAcorde*: recoge toda la lógica relacionada con la prueba del modo Adivinar Acorde, destacan los siguientes métodos:
    1. *Respuesta_seleccionada*: Se ejecuta cuando el usuario selecciona una opción, guardando su valor. En el primer nivel reproduce la opción seleccionada.
    2. *SeleccionaAcordesAleatorios*: Elige de manera aleatoria entre los acordes del parámetro recibido la respuesta correcta y la almacena.
    3. *ComprobarAcordes*: Sirve para corregir la prueba y modificar los colores para mostrar la solución. También actualiza la puntuación según corresponda y muestra el pop up de cambio de nivel si es necesario.

Además, la clase en cuestión presenta métodos más sencillos con funciones triviales para reproducir los sonidos, gestionar los ficheros de audio y repetir el nivel.

- Clase *SeleccionNivelAdivinarAcorde*: presenta de forma dinámica los niveles disponibles para el usuario para dicho modo de juego y establece el nivel seleccionado en el controlador. Muestra el pop up tutorial del modo de juego si es la primera vez del usuario.

- Clase *TutorialNivelAdivinarAcorde*: accesible a través del botón Ayuda disponible en el menú de seleccionar el nivel del modo Adivinar Acorde, muestra y reproduce todos los acordes posibles formados para la nota y octava que elija el usuario.

El subpaquete *Crear* contiene los siguientes métodos:

- Clase *CrearAcorde*: recoge toda la lógica relacionada con la prueba del modo Crear Acorde. Destacan los siguientes métodos:

    1. *Respuesta_seleccionada*: se ejecuta cuando el usuario selecciona una nota, añadiendo su valor al conjunto de notas seleccionadas. Excepto en el último nivel, reproducirá el acorde que el usuario ha formado hasta el momento con las opciones seleccionadas más la nota de inicio.
    2. *SeleccionaAcordesAleatorios*: selecciona el acorde correcto de manera aleatoria de entre los acordes recibidos del controlador y lo almacena en una variable.
    3. *SeleccionaNotasAleatorios*: genera una serie de notas aleatorias que más tarde se usaran como opciones de la prueba, usando una serie de limitaciones basadas en el nivel y la nota inicial del acorde a formar.
    4. *ComprobarCrearAcordes*: comprueba si las notas elegidas coinciden con el acorde solicitado, actualiza la puntuación según corresponda y muestra el pop up de cambio de nivel si es necesario.

    Además, la clase en cuestión presenta métodos más sencillos con funciones triviales para reproducir los sonidos, gestionar los ficheros de audio y repetir el nivel.

- Clase *SeleccionNivelCrearAcorde*: presenta de forma dinámica los niveles disponibles para el usuario y establece el nivel seleccionado en el controlador una vez que el usuario elige. En caso de que sea la primera vez que el usuario ingresa en el modo de muestra el pop up tutorial del modo de juego.

- Clase *TutorialNivelCrearAcorde*: Accesible a través del botón Ayuda disponible en el menú de seleccionar el nivel del modo Crear Acorde. Muestra las notas que forman los acordes para cada una de las notas de inicio.

5.6.5 – Paquete *Notas*

El paquete Notas agrupa toda la lógica relacionada con el modo Notas de la aplicación. Este paquete está formado por las siguientes clases:

- Clase *AdivinarNota*: Implementa los métodos necesarios para la prueba de adivinar nota, entre los métodos más destacados se encuentran:

    1. *Respuesta_seleccionada*: se ejecuta cuando el usuario selecciona una opción, guardando su valor en la variable local correspondiente. En el primer nivel reproduce la opción seleccionada por el usuario.
    2. *ComprobarResultado*: comprueba que la opción del usuario coincide con la nota solicitada, marcando el rojo la opción seleccionada por el usuario si es incorrecta o en verde si es correcta. Actualiza la puntuación como corresponda y muestra el pop up de cambio de nivel si es necesario.

    Además, la clase presenta métodos más sencillos con funciones para reproducir sonidos, modificar la interfaz gráfica en función de las acciones del usuario y obtener el path de los ficheros de audio.

- Clase *SeleccionNivelAdivinarNota*: presenta de forma dinámica los niveles disponibles para el usuario y establece el nivel seleccionado en el controlador. Muestra el pop up de tutorial si es la primera vez que el usuario accede a este modo de juego.

- Clase *TutorialNivelAdivinarNota*: gestiona la pantalla que se origina al pulsar el botón Ayuda disponible en el menú de seleccionar el nivel. Permite al usuario escuchar todas las notas, eligiendo él la octava que se reproduce.

5.6.6 – Paquete *Intervalos*

El paquete Intervalos agrupa toda la lógica relacionada con el modo Intervalos de la aplicación, tanto el modo Adivinar Intervalo como el modo Crear Intervalo.

El paquete está formado por las siguientes clases y subpaquetes:

- *Clase SeleccionarModoIntervalos*: sirve de puente entre los dos modos de juego Adivinar Intervalo y Crear Intervalo. También se encarga de mostrar la imagen del rango del usuario en cada modo.

- *Clase TutorialIntervalos*: Accesible a través del botón Ayuda disponible en el menú de seleccionar el modo, muestra y reproduce todos los intervalos posibles sobre la nota LA4, junto con la diferencia en semitonos.

El subpaquete *Adivinar* se encarga del modo de juego *Adivinar y* contiene las siguientes clases:

- Clase *AdivinarIntervalo*: Recoge toda la lógica relacionada con la prueba del modo Adivinar Intervalo, destacan los siguientes métodos:

1. *Respuesta_seleccionada*: guarda la opción seleccionada por el usuario y pone visible el botón de comprobar.
2. *ComprobarIntervalos*: se encarga de comprobar si el intervalo seleccionado coincide con la solución. Actualiza la puntuación del usuario y muestra los pop ups de subida de nivel y rango si es necesario.
3. GetIntervaloConDif: Recibe una diferencia entre dos tonos y devuelve el intervalo al que corresponde.

Además, la clase en cuestión presenta métodos más sencillos con funciones triviales reproducir los intervalos y las notas, modificar la interfaz gráfica y reiniciar la actividad.

- Clase *SeleccionNivelAdivinarIntervalos*: muestra de forma dinámica los niveles disponibles para el usuario para dicho modo de juego y establece el nivel seleccionado en el controlador. También muestra el pop up tutorial del modo de juego si es la primera vez que el usuario ingresa en el modo.

El subpaquete *Crear* se encarga del modo de juego Crear y contiene las siguientes clases:

- Clase *CrearIntervalo*: Recoge toda la lógica relacionada con la prueba del modo Crear Intervalo, destacan los siguientes métodos:

    1. *OnCreate*: establece la nota de inicio del intervalo a crear y crea de forma dinámica las opciones de entre las notas posibles que genera el método *generaNotasAleatorias*.
    2. *Respuesta_seleccionada*: guarda la opción seleccionada por el usuario y pone visible el botón comprobar. En los primeros niveles reproduce la opción seleccionada.
    3. *SeleccionaNotasAleatorios*: genera un conjunto de posibles notas que podrían formar cualquier intervalo sobre la nota de inicio de un intervalo recibido, aplicando las restricciones del nivel.
    4. *ComprobarResultado*: comprueba que la opción seleccionada coincide con el intervalo solicitado. Actualiza las puntuaciones y estadísticas como corresponda.
    4. GetIntervaloConDif: recibe la diferencia entre dos notas y devuelve el intervalo al que corresponde.

    Además, la clase en cuestión presenta métodos más sencillos con funciones triviales reproducir los intervalos y las notas, modificar la interfaz gráfica y reiniciar la actividad.

- Clase *SeleccionNivelCrearIntervalo*: presenta de forma dinámica los niveles disponibles para el usuario y establece el nivel seleccionado en el controlador. Si es la primera vez que el usuario ingresa en el modo de juego muestra el pop up tutorial.

5.6.7 – Paquete *ImitarAudio*

El paquete *ImitarAudio* contiene todo lo referido al modo de juego *Entonación.* En este paquete podemos encontrar las siguientes clases:

- *SeleccionOctavasImitar:* gestiona la actividad de selección de rango vocal, establece el rango vocal del usuario para determinar el conjunto de notas que se solicita.
- *NivelesImitar:* gestiona la actividad para seleccionar el nivel dentro del modo de juego *Entonación,* así como el pop up de tutorial del modo.
- *NotasImitar*: clase auxiliar que contiene una nota, la octava en la que se encuentra y el número de veces que se detecta durante la grabación.
- *ReproducirImitar:* es la clase principal del paquete *ImitarAudio*, en esta clase se implementa toda la funcionalidad del modo de juego *Entonación.* Los métodos más importantes que contiene son:

    1. *RangoPorNivel:* adapta el rango vocal al nivel en el que se encuentra el usuario en *Entonar*.
    2. *InicializaPorDificultad:* adapta la dificultad del modo de juego en función del nivel.
    3. *Reproducir:* reproduce la nota a entonar en el ejercicio y modifica las reproducciones restantes del usuario.
    4. *Comparar:* compara la nota grabada por el usuario con la nota a entonar en el ejercicio y muestra el resultado al usuario. Por último, actualiza las estadísticas del usuario en la base de datos.
    5. *Contador:* realiza la cuenta atrás que se muestra al usuario antes de la grabación y llama al método *InicializaPitch* para iniciar la captación de frecuencias.
    6. *InicializaArrays:* inicializa todos los elementos de Pitch para que no afecten grabaciones anteriores.
    7. *InicializaPitch:* inicializa el proceso de grabación de frecuencias. Usando la clase *PitchDetectionHandler* del paquete de *TarsosDSP,* recoge las diferentes frecuencias que el micrófono detecta durante cinco segundos y se las pasa al método *hallaMax.*
    8. *HallaMax:* detecta la octava en la que se sitúa la frecuencia que recibe, comparándola con las frecuencias mínimas y máximas de cada octava. A continuación, pasa la frecuencia y la octava calculada al método *compruebaSiEsNota.*
    9. *CompruebaSiEsNota*: sitúa la frecuencia en uno de los semitonos, para determinar a qué nota corresponde. Si esa nota ya ha sido registrada anteriormente en la grabación, añade uno al contador de esa nota en concreto.
    10. *PoneNota:* una vez finalizada la grabación, este método comprueba cuál ha sido la nota que más veces ha entonado el usuario durante los cincos segundos de grabación. También calcula el porcentaje de afinación comparando la media de frecuencias de la nota predominante con la frecuencia perfecta de la nota en la octava determinada.

5.6.8 – Paquete *Ritmos*

El paquete *Ritmos* contiene todo lo referido a los modos en el apartado *Ritmos* estos modos son *Dibujar Ritmos* e imitar *ritmos*. Podemos encontrar diversas clases y subpaquetes:

- *SeleccionNivelDibujarRitmos:* sirve de puente entre los dos modos de juego y muestra qué rango tiene el usuario en cada uno.

- *MediaPlayerRitmos:* se usa para coordinar y gastar los mínimos recursos posibles a la hora de usar la clase *MediaPlayer* de *Android*. Esta clase contiene los métodos *init* que inicializa los *mediaPlayers*, *play* que reproduce el audio cargada y *stop* que para y libera todos los *mediaPlayers* para así disminuir el impacto. En esta clase cargamos hasta los 4 instrumentos posibles junto con el metrónomo.

A partir de aquí tenemos dos subpaquetes: *Hallar* y *Crear.* Ambos paquetes contienen dos clases prácticamente iguales que son *SeleccionNivelDibujarRitmos* y *SeleccionNivelImitarRitmos* para seleccionar los niveles de los modos *Dibujar Ritmos* e *Imitar Ritmos* y mostrar sus tutoriales la primera vez que accedemos a ellas.

El paquete *Hallar* además contiene la clase *DibujarRitmo* que es la clase principal del modo de juego *Dibujar Ritmos* y principalmente contiene los siguientes métodos:

1. *Play:* activa el hilo [26] de reproducción del ritmo propuesto para el ejercicio.
2. *Pause:* pausa el hilo de reproducción en el mismo instante que se ejecuta, pudiéndose poner en marcha más tarde desde ese punto.
3. *Para:* para el hilo de reproducción del ritmo propuesto, empezando desde el principio si se ejecuta el método *play.*
4. *Stop:* este método corrige si el ritmo que ha dibujado el usuario corresponde con el ritmo propuesto. También actualiza la información de la base de datos del usuario en ese modo de juego.
5. *AgregaFigura:* es un método auxiliar que utilizamos para rellenar el ritmo.

El paquete *Crear* además contiene la clase *ImitarRitmo* que es la clase principal del modo de juego *Imitar Ritmos* y principalmente contiene los siguientes métodos:

1. *Play:* activa el hilo de reproducción del ritmo e inicia la línea del tiempo.
2. *ReproducirRitmosPropio:* activa el hilo de reproducción del ritmo que ha creado el usuario.
3. *BorrarRitmoPropio:* borra el ritmo que ha creado el usuario.
4. *Para:* para los hilos de reproducción de los ritmos.
5. *Comprobar:* comprueba si el ritmo imitado por el usuario y el ritmo propuesto son iguales, modificando la interfaz.

6. *MostrarSolucion, AddBotonesSolucion, PintaFiguraInvisible y PintaFigura:* métodos auxiliares utilizados en el método *Comprobar* que muestra al usuario la respuesta correcta por medio de iconos con color.
7. *CompruebaArrays:* compara los sonidos de cada instrumento introducido por el usuario con los solicitados.
8. *RegistraPalmada, RegistraTambor, RegistraPlatillo y RegistraCaja:* registran instrumentos en el ritmo del usuario en el instante de tiempo indicado por la línea del tiempo.

5.6.9 – Paquete *Mix*

En este paquete se encuentran las clases que implementan el modo *Mix*. Se compone de dos clases nuevas para realizar el control de la interfaz gráfica y un subpaquete con las clases que representan cada tipo de prueba.

- La clase *SeleccionNivelMix* se encarga de controlar la interfaz de usuario de la pantalla de selección de nivel, así como de sincronizar la aparición de las distintas pruebas. Una vez que el usuario ha seleccionado un nivel, se comunica con la clase *ControladorMix* para que establezca el nivel del modo, y comienza la prueba. Para controlar el desarrollo del modo, se emplean dos funciones:

  1. S*iguienteEjercicio*: se encarga de comprobar si el *Mix* ha finalizado. Si no es así, solicita a la clase *ControladorMix* la prueba a lanzar y la inicia. Si ha finalizado, llama al método de la clase *ControladorMix* que se encarga de establecer el resultado final y a continuación inicia la actividad *ResultadoMix.*
  2. *OnActivityResult*: debido a que esta clase hereda de la clase *Activity*, para mantenerla a la espera de que finalice cada prueba para lanzar la siguiente es necesario lanzar las pruebas con el método *startActivityOnResult().* Esto conlleva que al acabar la actividad que inicia recibe un parámetro de resultado. Este método es el encargado de recibir ese parámetro, que coincide con el resultado de la prueba. Si la prueba ha finalizado sin que el usuario la haya abortado, se informa al *ControladorModoMix* del resultado y se avanza al siguiente ejercicio.

- La clase ResultadoMix se encarga de controlar la interfaz de usuario de la pantalla de resultados de la prueba. Una vez que obtiene los resultados por parte de la clase ControladorMix, comprueba si el número de aciertos es igual o superior al requerido para superar la prueba. En función de esto, modifica el mensaje y el color del texto que informa del resultado del modo.

El subpaquete *Ejercicios* contiene todas las clases que implementan modificaciones de las pruebas originales. Para ello cada clase extiende la clase original, modificando dos métodos y dejando los restantes sin sobrescribir:

1. Método *onCreate*: después de llamar al método de la clase de la que hereda, se añade un índice para mostrar al usuario el progreso dentro del modo Mix.

2. Métodos de corrección: cada clase tiene su propio método para establecer el resultado de la prueba. En este subpaquete, se sobrescribe esos métodos para evitar inserciones en la base de datos de manera individual y se añade un objeto *Intent* para informar a la clase *SeleccionNivelMix* del resultado de la prueba.

5.7 – IMPLEMENTACIÓN DEL SISTEMA DE RANGOS Y PUNTUACIONES

Nuestra aplicación implementa un sistema de niveles para cada modo de juego, de forma que cada uno cuenta con desde 6 hasta 10 niveles diferentes, según el modo, accesibles cada uno de ellos tras alcanzar cierta puntuación en el nivel anterior. En cada uno de ellos se modificará una serie de parámetros que transformará la experiencia de usuario para cada nivel.

El sistema funciona a través de la clase Controlador, explicada anteriormente. Allí se guardan los de parámetros de dificultad que varían según el modo en cuestión y establecidos por la aplicación cuando el usuario selecciona un nivel. Dichos parámetros son recuperados por la clase correspondiente del modo, usándolos para aumentar o disminuir la dificultad.

El sistema de niveles es complementado con un sistema de puntuación implementado a partir de rangos para cada modo de juego, ideado como incentivo al usuario para subir y mantener el nivel ante la creciente dificultad, proporcionándole una razón para mejorar. Nuestra aplicación presenta hasta 7 rangos distintos, ascendentes en su progresión con el siguiente orden: Principiante, Aprendiz, Veterano, Experto, Gran Maestro y Leyenda, apoyados todos ellos en una puntuación global, única para para cada modo y representados por una figura musical: semifusa, fusa, semicorchea, corchea, negra y blanca, en ese orden ascendente en la progresión. Dicha figura se mostrará a la derecha del botón de acceso a cada modo indicando al usuario el rango en el que se encuentra.

Para cada modo de juego el usuario comienza en el rango Principiante con la puntuación 0 en todos los modos de juego. A continuación, deberá aumentar su rango, subiendo su puntuación en un modo. En los primeros niveles cada prueba acertada aumentará la puntuación en tres puntos y cada prueba fallada la disminuirá en un punto; en los niveles intermedios cada prueba fallada restará dos puntos y en los últimos niveles restará 1 punto sobre la puntuación total del modo, todo ello siempre y cuando el nivel de la prueba sea el último disponible.

El usuario subirá de rango tras alcanzar cierta puntuación en el modo, dicho umbral de puntuación cambiará según el modo, ya que cada uno cuenta con un número distinto de niveles.

La subida de rango suele corresponder al desbloqueo de un nuevo nivel, no obstante, eso no siempre es así, ya que hay ciertos modos donde, debido a un número de niveles superior al número de rangos disponibles, puede haber varios niveles para cada rango.

La puntuación que le queda al usuario para alcanzar el siguiente nivel se muestra debajo del botón para acceder al nivel actual.

5.8 – IMPLEMENTACIÓN DEL ANÁLISIS Y REPRODUCCIÓN DEL AUDIO

Consideramos que el audio es la base de *BasicMusic* ya que al fin y al cabo es la base de la música, por lo tanto, quisimos que nuestro análisis y reproducción de este fuese algo en lo que se pueda confiar y que el usuario no tuviera ninguna duda de que los resultados que éste obtenía fuesen cien por cien veraces.

El análisis lo utilizamos, sobre todo, en el modo de juego *Entonación* ya que es el único modo de juego en el que utilizamos audio externo a la aplicación. La parte central de este análisis lo realizamos con ayuda de la librería *TarsosDSP:* esta librería analiza audio a bajo nivel y permite realizar multitud de tipos de análisis de audio en *Android,* aunque el único tipo de análisis que utilizamos nosotros es el análisis de frecuencias. Debido a que, a grandes rasgos, las notas musicales son frecuencias, por ejemplo, un *LA* de cuarta octava siempre tiene una frecuencia de alrededor de 440 hercios, y si multiplicas o divides esa frecuencia entre dos, se consigue la frecuencia de los *LA* de octavas superiores o inferiores, (*LA* de tercera octava 220 hercios, *LA* de quinta octava 880 hercios), este tipo de análisis nos proporciona todo lo que necesitamos para hacer un ejercicio de entonación fiable.

Además de *TarsosDSP*, necesitamos un método para asegurar que el sonido captado por el micrófono era el sonido que el usuario quería que la aplicación captase, y no sonidos residuales que el usuario pudiese tener a su alrededor. Para esto utilizamos clases del propio *Android*, concretamente del paquete *AudioFX [27]*: *AcousticEchoCanceler, AutomaticGainControl, NoiseSuppressor.* Estas clases ayudan a que el audio que capta la aplicación sea el audio deseado, aunque, el hecho de que estas clases funcionen correctamente depende de las características técnicas del terminal dónde se esté ejecutando la aplicación.

En cuanto a la reproducción del audio, la clase principal que utilizamos es una clase del propio *Android* del paquete *Media*, *MediaPlayer [28].* Esta clase permite reproducir a través del dispositivo, un archivo dentro del proyecto. En este caso nosotros hemos usado archivos *WAV [29]* que son todas las notas musicales que se pueden tocar en un piano de cola. Los archivos *WAV* los obtuvimos de la web de *FREEPATS* que ofrecen archivos *WAV* de varios instrumentos gratis

CAPÍTULO 6 – FUNCIONALIDAD

La aplicación contiene tres modos de juego principales: adivinar audio, Entonación y ritmos.

6.1 – ADIVINAR AUDIO

En este modo de juego, el usuario debe reconocer una muestra de audio que reproduce el dispositivo y responder a la pregunta que se le realiza. Para la reproducción del audio se emplean archivos que contienen las notas musicales de un piano.

Este modo se subdivide en tres categorías: notas, intervalos y acordes.

- Notas

La aplicación elige una nota de un conjunto y el usuario debe adivinar de qué nota se trata. El usuario dispone de un botón para reproducir la nota a adivinar. Contiene un nivel de ayuda en el cual el usuario puede escuchar todas las notas seleccionando la octava a reproducir.

- Intervalos

Se ofrecen dos modos: adivinar intervalo y crear intervalo.

  - Adivinar intervalo

La aplicación elige un intervalo de un conjunto y el usuario debe adivinar de qué intervalo se trata marcando la opción con el nombre correcto. El usuario dispone de un botón para reproducir el intervalo a adivinar.

  - Crear intervalo

La aplicación elige un intervalo de un conjunto y una nota sobre la que se inicia el intervalo y el usuario tiene que escoger la segunda nota que completa el intervalo. En este modo de juego el intervalo no se reproduce, sino que se le muestra al usuario en forma de texto. Respecto a la nota de inicio, el modo contiene un botón para reproducirla.

- Acordes

Se ofrecen dos modos: adivinar acorde y crear acorde.

  - Adivinar acorde

La aplicación selecciona un acorde y el usuario debe acertar cuál es el acorde correcto entre las distintas opciones que se le ofrecen. Para ello, dispone de un botón para reproducir el acorde a acertar.

- Crear acorde

La aplicación selecciona un acorde y la nota de inicio, que puede reproducir mediante un botón. El usuario tiene que seleccionar todas las notas restantes para construir el acorde.

6.2– ENTONACIÓN

En este modo el usuario debe tratar de entonar la nota que le proponemos en cada ejercicio. Esta nota se puede reproducir pulsando un botón. Tras identificar la nota, el usuario debe pulsar el botón *Grabar Muestra,* y tras una cuenta atrás el dispositivo comenzará a grabar y analizar las frecuencias de todos los sonidos que capte durante cinco segundos. *BasicMusic* mostrará la nota que más ha captado durante ese periodo y si el usuario cree que es la nota correcta, puede pulsar el botón comprobar haciendo que el sistema compare la nota propuesta y la nota entonada para ver si se corresponden. En caso de acertar mostramos al usuario mediante un porcentaje lo cerca que ha estado de la entonación perfecta.

Según el usuario vaya escalando niveles, la dificultad va aumentando. Los factores que varían son: la cantidad de veces que usuario puede reproducir la nota propuesta, la cantidad de intentos que tiene el usuario al comparar la nota entonada y la propuesta y el rango de notas propuestas. Además, en caso de los niveles más bajos, damos por buenas notas que se correspondan, pero no estén en la misma octava.

6.3 – RITMOS

Nuestro objetivo con los modos de juego que se corresponden a *Ritmos* es mejorar la escucha e interpretación de ritmos por parte del usuario. Podemos dividir este campo en los siguientes modos de juego.

- *Dibujar Ritmos*

En este modo el objetivo del usuario es intentar “dibujar” el ritmo que proponemos con las líneas de botones que mostramos.

El usuario tiene un control completo para reproducir, parar y pausar el ritmo propuesto con los botones de la cabecera.

A continuación de esos botones se encuentran unas “líneas de tiempo”, nuestra intención con esta línea de botones es representar un compás cuatro por cuatro, como explicamos en el apartado *Teoría*. En nuestro caso la unidad de medida mínima que usamos

es la semicorchea, es decir, que podemos subdividir el compás en hasta dieciséis partes (en niveles bajos subdividimos el compás en ocho para facilitar el ejercicio). Para realizar esta subdivisión utilizamos botones. Cada uno representa un instante de tiempo dentro del compás. Si el usuario cree que en ese instante de tiempo suena un sonido, debe marcar el botón, que cambia de color.

Por último, una vez que el usuario haya acabado de "dibujar" el ritmo, debe pulsar el botón *Comprobar* para que el sistema compare el ritmo dibujado con el ritmo propuesto e indique al usuario si su ritmo es correcto o incorrecto. En caso de fallo, el sistema corregirá el dibujo indicando al usuario dónde ha fallado.

- *Imitar Ritmos*

En este modo el usuario tiene que intentar imitar el ritmo propuesto mediante la pulsación de botones que representan instrumentos en tiempo real.

Como en el modo *Dibujar Ritmos* el usuario puede encontrar una cabecera de botones para poder pausar y reproducir el ritmo propuesto. Más abajo se encontrará con una "línea de tiempo" cuyo funcionamiento es idéntico al del anterior modo, salvo que en este caso el usuario no puede pintar sobre ella y simplemente es algo simbólico, representando el avance del ritmo.

Cuando el ritmo se está reproduciendo se activan los botones de los instrumentos para que el usuario pueda ir imitando el ritmo propuesto pulsándolos. Cuando se pulse un botón de un instrumento se añadirá el sonido de ese instrumento en el instante indicado por la línea de tiempo.

Justo debajo de los botones de instrumentos el usuario puede encontrar un grupo de botones que sirven para controlar el ritmo que ha ido imitando pudiéndolo reproducir para comprobar si es correcto.

Pulsando el botón comprobar el sistema comparará el ritmo propuesto con el ritmo imitado, indicando al usuario si su ritmo es correcto o incorrecto. En caso de fallo la aplicación indicará al usuario dónde ha fallado o le ha faltado algún instrumento, mediante iconos.

6.4 – INTERFAZ DE USUARIO

6.4.1 – Introducción

A lo largo del desarrollo de la aplicación, la interfaz de usuario ha cambiado sustancialmente tiempo debido al feedback recibido por parte del director, personas a las que se les han proporcionado distintas versiones y nuestra propia experiencia.

Tras las distintas revisiones, las conclusiones que fuimos sacando para realizar la versión final fueron las siguientes:

- La interfaz de usuario debía ser lo más simple e intuitiva posible debido a la naturaleza de la aplicación. El usuario no tendría por qué tener conocimientos musicales previos para poder utilizarla.
- En un primer momento se dejaba en manos del usuario el ajuste de los parámetros de los niveles. Esto suponía que para comenzar un nivel el usuario debía varias pantallas de configuración, lo que ralentizaba el uso.
- En algunos modos de juego se necesitaban introducir más ayudas dentro del propio nivel que ayudasen al usuario a la hora de decidirse por una respuesta.
- Resultaba conveniente introducir pantallas para que el usuario pudiese practicar los conceptos que debía poner en práctica antes de realizar las pruebas.
- Se realizaron diferentes pantallas de tipo pop up para guiar al usuario en el uso de la aplicación, explicando los diferentes controles de cada modo de juego.
- Resultaba interesante algún modo de juego que consistiera en la mezcla de todos los demás.
- El usuario debería ser informado de algún modo cuando desbloquea nueva funcionalidad o se reduce el nivel de dificultad máxima en el que puede jugar.
- En general, la interfaz era bastante repetitiva visualmente. Resultaba conveniente introducir algún botón con un estilo diferente o el uso de imágenes en lugar de texto dentro de ellos.
- Para mejorar la dinámica de la aplicación, era interesante introducir un botón en todos los modos de juego que permitiera repetir el nivel sin tener que volver atrás.

Finalmente, el conjunto de las pantallas y la funcionalidad de cada una de ellas ha quedado del siguiente modo.

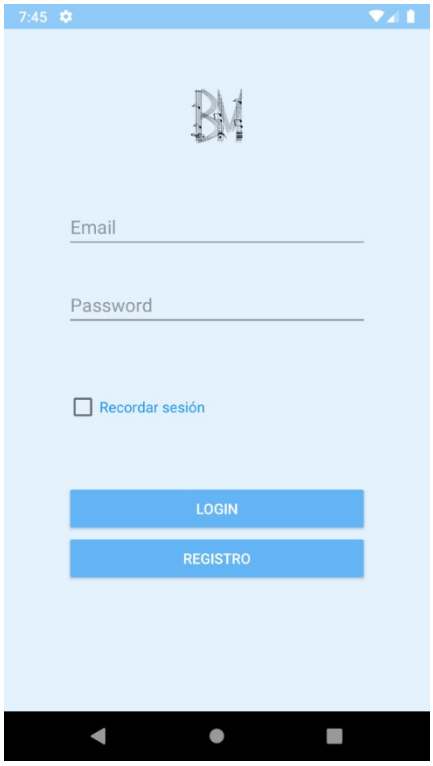
6.4.2 – Pantallas de la aplicación

6.4.2.1 – Pantalla login/registro

Las pantallas de login y registro de nuestra aplicación son las típicas pantallas que hay en prácticamente todas las aplicaciones.

En cuanto al registro, el usuario debe introducir un correo electrónico que va a servir para identificarse dentro de nuestra aplicación, además de su nombre y una contraseña. En caso de que el correo electrónico ya existiese en nuestra base de datos se muestra un mensaje de error al usuario, así como si no introdujera un email correctamente formado.

En la pantalla de login pedimos al usuario su email y su contraseña. Además, hemos añadido una opción de recordar sesión para que el usuario no tenga que volver a hacer login una vez abandone la aplicación.



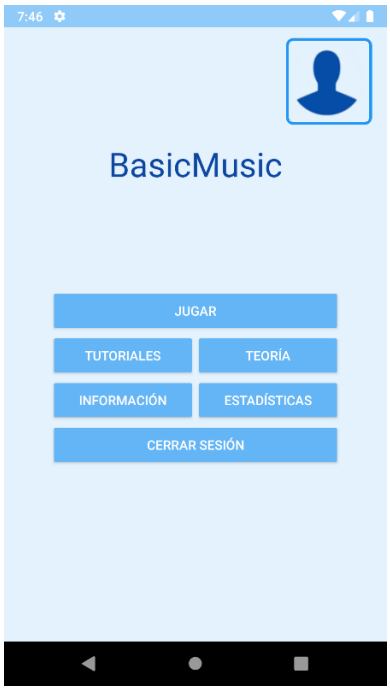
Pantalla de Login

Pantalla de Registro

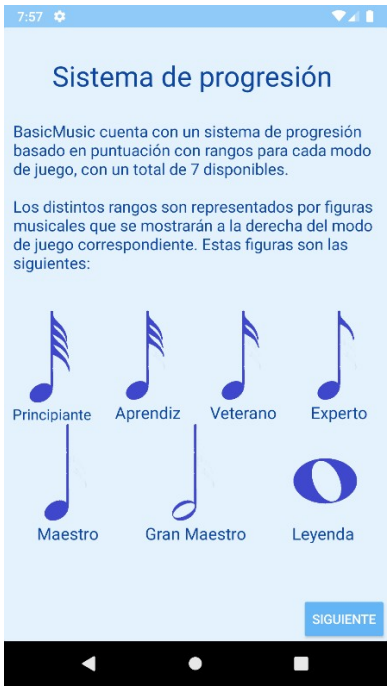
6.4.2.2 – Pantalla de inicio

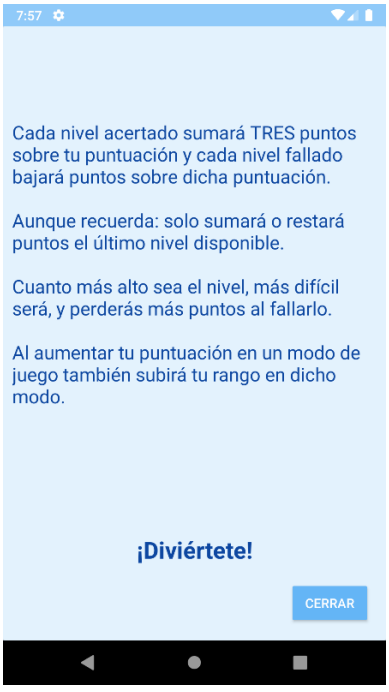
Se trata de la pantalla principal, aquella que se muestra al usuario al iniciar la aplicación con su cuenta. En ella se muestra el nombre de la aplicación, una serie de botones que conducen a distintas funcionalidades y en la esquina superior derecha un botón con la imagen de una silueta humana para editar la información del perfil.

Cuando un usuario ingresa por primera vez a la pantalla de inicio de la aplicación se muestra dos pop up diferentes y contiguos que explican el funcionamiento del sistema de progresión implementado en la aplicación.



Pantalla de Inicio





Tutorial de puntuaciones

6.4.2.3 – Pantalla de editar perfil

La pantalla de editar perfil tiene como objetivo proporcionar al usuario una forma de modificar los datos principales de su cuenta. En ella, el usuario puede cambiar su nombre y la contraseña, teniendo que confirmar dicha contraseña.

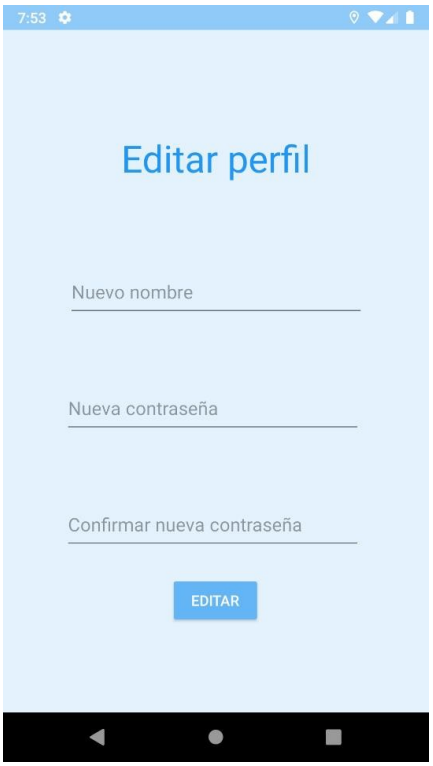
Si las contraseñas introducidas por el usuario no coinciden, un mensaje de error en la parte superior de la pantalla.

6.4.2.4 – Pantallas de tutoriales

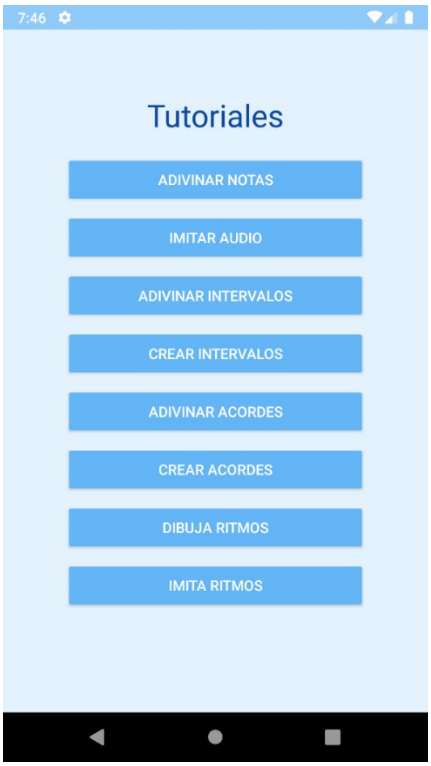
Las pantallas de tutoriales sirven para mostrar al usuario cómo se realiza a cada uno de nuestros modos de juego. Se muestra automáticamente cuando el usuario accede a un modo por primera vez.

Estas pantallas consisten en una recreación de las pantallas de los modos de juego, en las que van apareciendo los controles y su explicación de manera progresiva. Los botones "*Siguiente*" y "Anterior" sirven para navegar entre las pantallas del tutorial en cuestión.

Además de mostrarse solo la primera vez que se va a jugar a un modo de juego, se pueden revisar en cualquier momento desde el botón "*Tutoriales*" de la pantalla principal.



Pantalla de Editar Perfil



Pantalla de menú de tutoriales

6.4.2.5 – Pantalla de teoría

Debido a que el usuario no tiene por qué tener ningún conocimiento previo sobre teoría musical, decidimos introducir una pantalla que explicase los conceptos mínimos que necesita tener

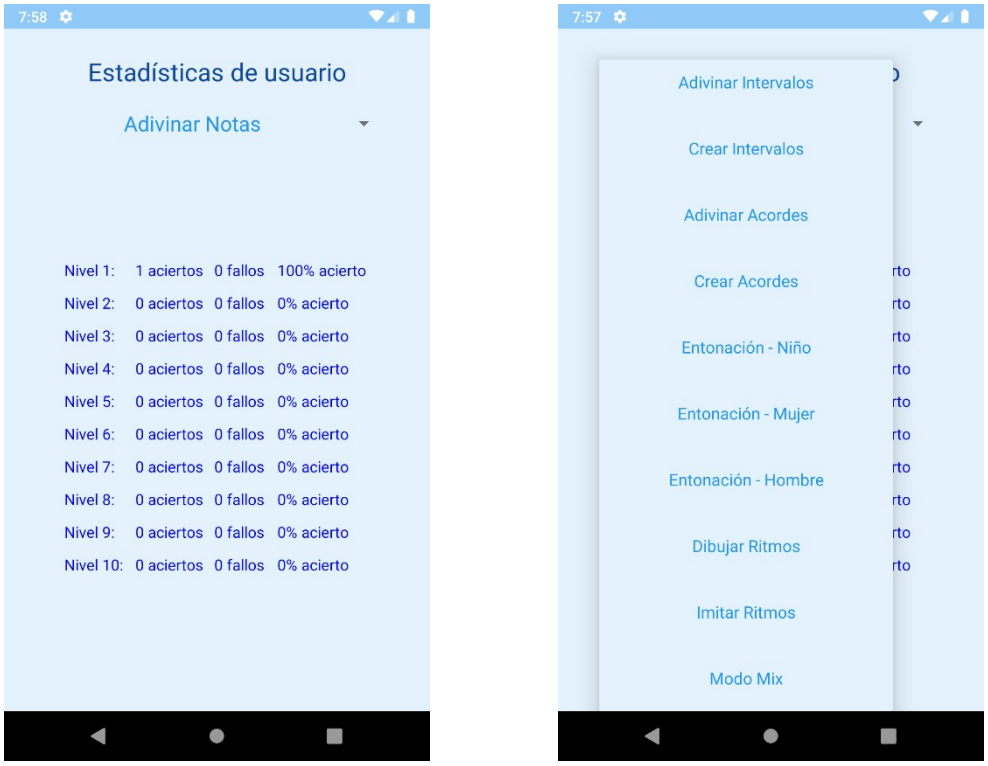
una persona que quiera utilizar la aplicación para comprender los distintos modos de juego. Esta pantalla está compuesta por dos vistas diferentes, cada una con distintos conceptos musicales.

Pantallas de Teoría

6.4.2.6 – Pantalla de estadísticas

La pantalla de estadísticas es la encargada de mostrar la información almacenada en la base de datos, de manera que el usuario pueda ver su progreso en los diferentes modos de juego de la aplicación.

El usuario puede seleccionar el modo del que desea ver las estadísticas mediante una lista desplegable. Para el modo de entonación se muestra el número de intentos y el porcentaje de afinación por nivel, mientras que para el resto de los modos el usuario puede ver su número de aciertos y su número de fallos por nivel, así como el porcentaje de veces que lo ha superado.



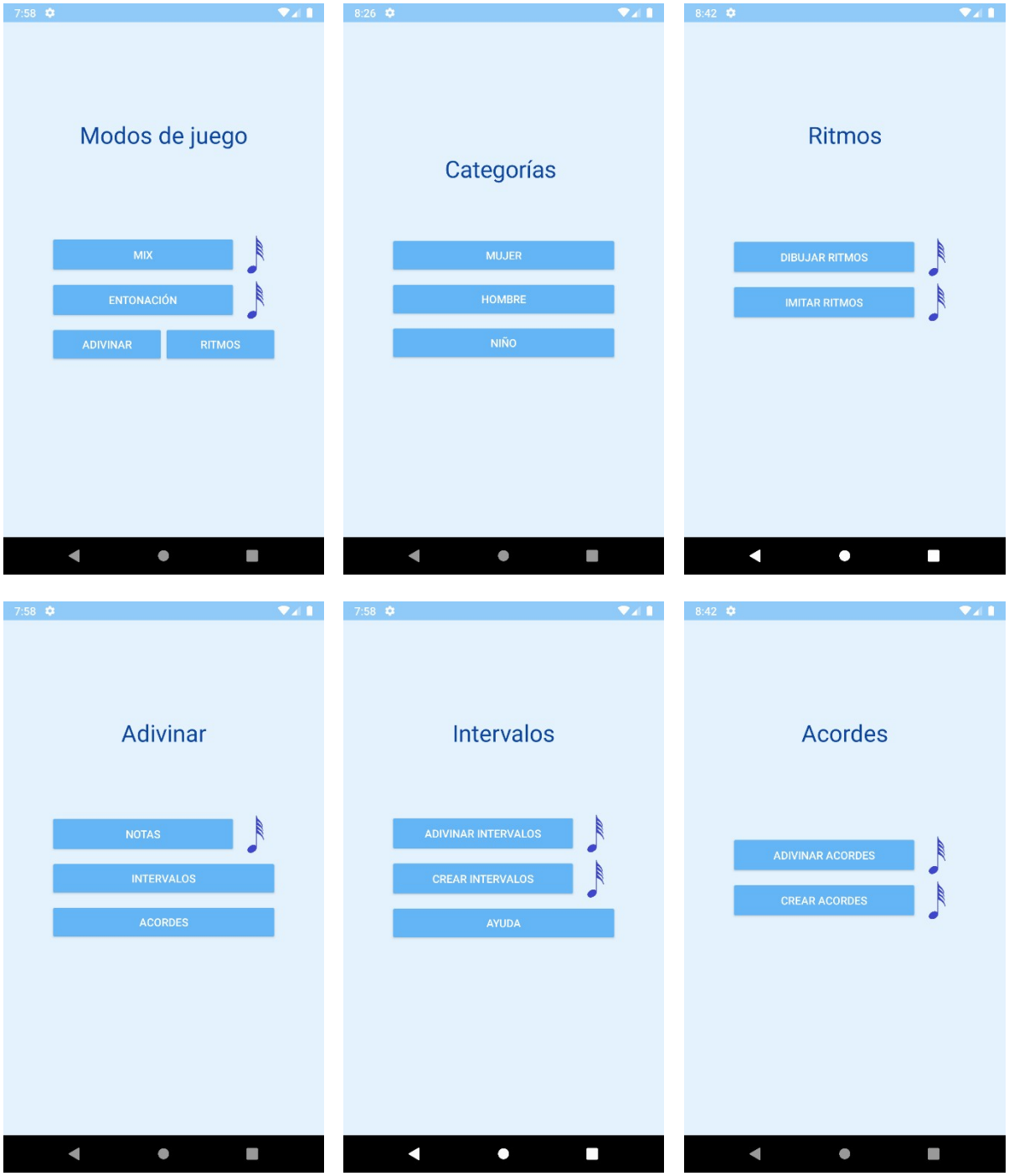
Pantalla de Estadísticas 1

6.4.2.7 – Pantalla de selección del modo de juego

Las pantallas de selección de modo de juego se muestran una vez que el usuario pulsa sobre el botón de jugar de la pantalla principal. En ellas se pueden diferenciar dos tipos de botones:

- Botones que dirigen a la selección del nivel del modo de juego. A la derecha del botón se puede visualizar el icono correspondiente al rango que tiene el usuario en dicho nivel.
- Botones que dirigen a otras pantallas de selección. Estos botones no muestran ningún icono y al pulsarlos redirigen a otras pantallas más específicas de selección.

Cada pantalla de selección de modo se especifica en su propio fichero XML y supone una Activity independiente.



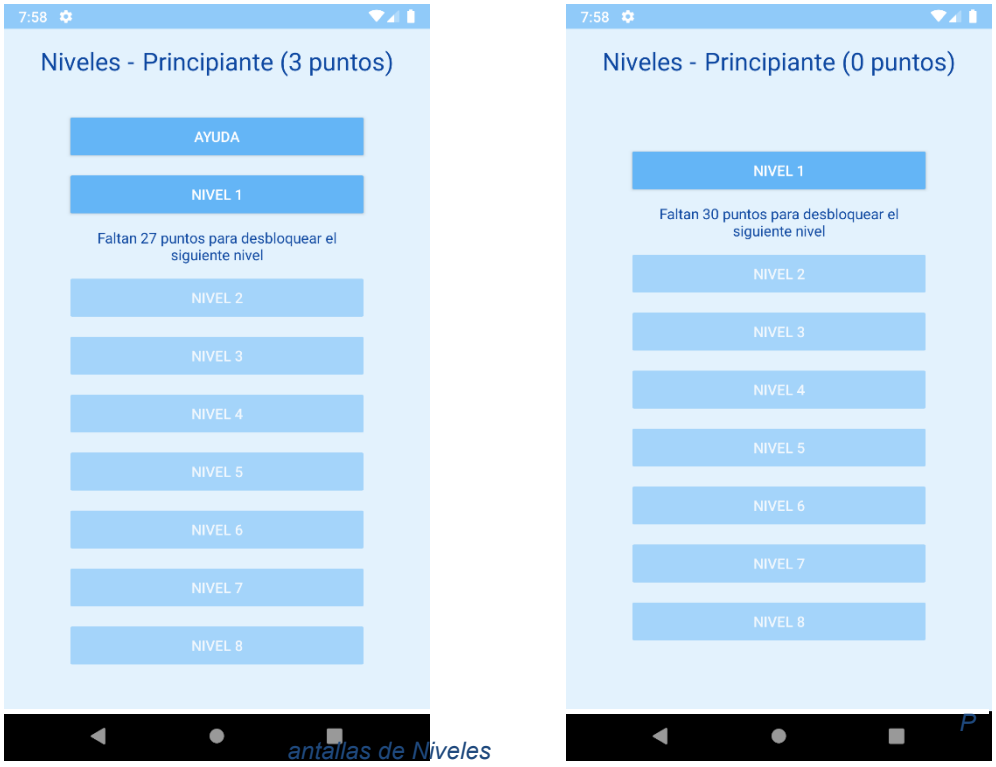
Pantallas de menú

6.4.2.8 – Pantalla selección del nivel

Al seleccionar un modo de juego, se muestra esta pantalla en la que el usuario puede elegir el nivel que quiere realizar mediante una lista de botones. Toda la interfaz es común a todos los modos de juego, adaptandose de manera dinámica [24] en función de la cantidad de niveles que tenga cada modo.

Dependiendo de la puntuación algunos niveles se encuentran bloqueados. En ese caso, se muestra también al usuario el número de puntos que necesita para desbloquear el siguiente nivel. Como característica adicional, algunas pantallas de nivel contienen un botón de *Ayuda* (explicados en el apartado 5.5.2.15).

Como característica adicional, todas las pantallas de nivel que corresponden a los modos incluidos en la categoría Adivinar contienen un botón adicional de Ayuda.



Pantallas de niveles

6.4.2.9 – Pantallas modo acordes

Existen dos pantallas principales dentro del modo acordes: una para el modo *Crear Acordes* y otra para el modo *Adivinar Acordes*.

Las dos pantallas incluyen el enunciado en forma de texto, seguido de un botón que reproduce la nota/acorde en función del modo.

A continuación, aparecen las distintas opciones de respuesta (notas que forman el acorde pedido o acordes posibles, según el modo). La/s opción/es seleccionadas se ensalzarán en un tono azul oscuro una vez seleccionadas.

En la parte inferior, las pantallas muestran el botón de Comprobar, que solo aparece cuando se selecciona una posible respuesta. Al pulsarlo, se resaltan en verde la/las opciones correctas y en rojo la/las opciones seleccionadas que sean incorrectas

Por último, en los primeros niveles se encuentra el botón de "*Info*", que lleva al usuario a una ventana de ayuda relacionada con la prueba que está realizando.

Pantallas de Crear Acorde *Pantallas de Adivinar Acorde*

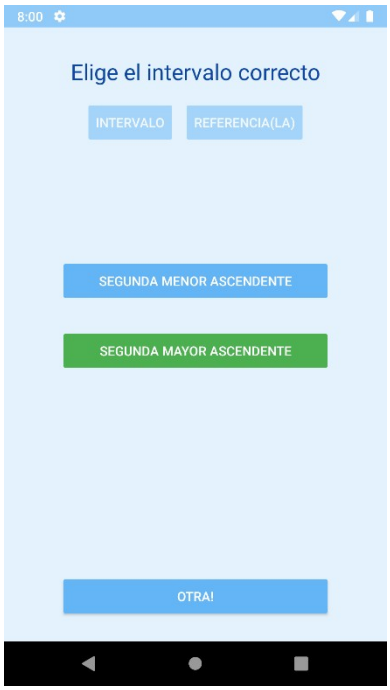
6.4.2.10 – Pantallas modo intervalos

Existen dos pantallas del modo intervalos: una para el modo Crear Intervalos y otra para el modo Adivinar Intervalos.

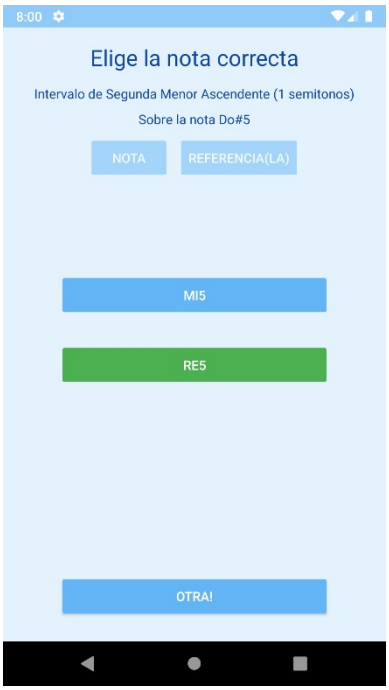
En el caso de *Adivinar Intervalo*, se muestra un botón que reproduce el intervalo a adivinar, mientras que en *Crear Intervalo* el enunciado está en forma de texto. Ambos modos incluyen otro botón que reproduce una nota referencia (LA4) para ayudar al usuario.

Seguidamente aparecen las distintas opciones (notas que forman el intervalo pedido o intervalos posibles, según el modo), que oscurecen su tono al ser seleccionadas.

En la parte inferior, las pantallas muestran el botón de "*Comprobar*" al seleccionar una opción. Una vez pulsado, comprueba la respuesta, resaltando en verde la/las opciones correctas y en rojo la/las opciones seleccionadas que sean incorrectas



Pantalla de Adivinar Intervalo



Pantalla de Crear Intervalo

6.4.2.11 – Pantallas del modo ritmos

Hay dos pantallas del modo ritmos.

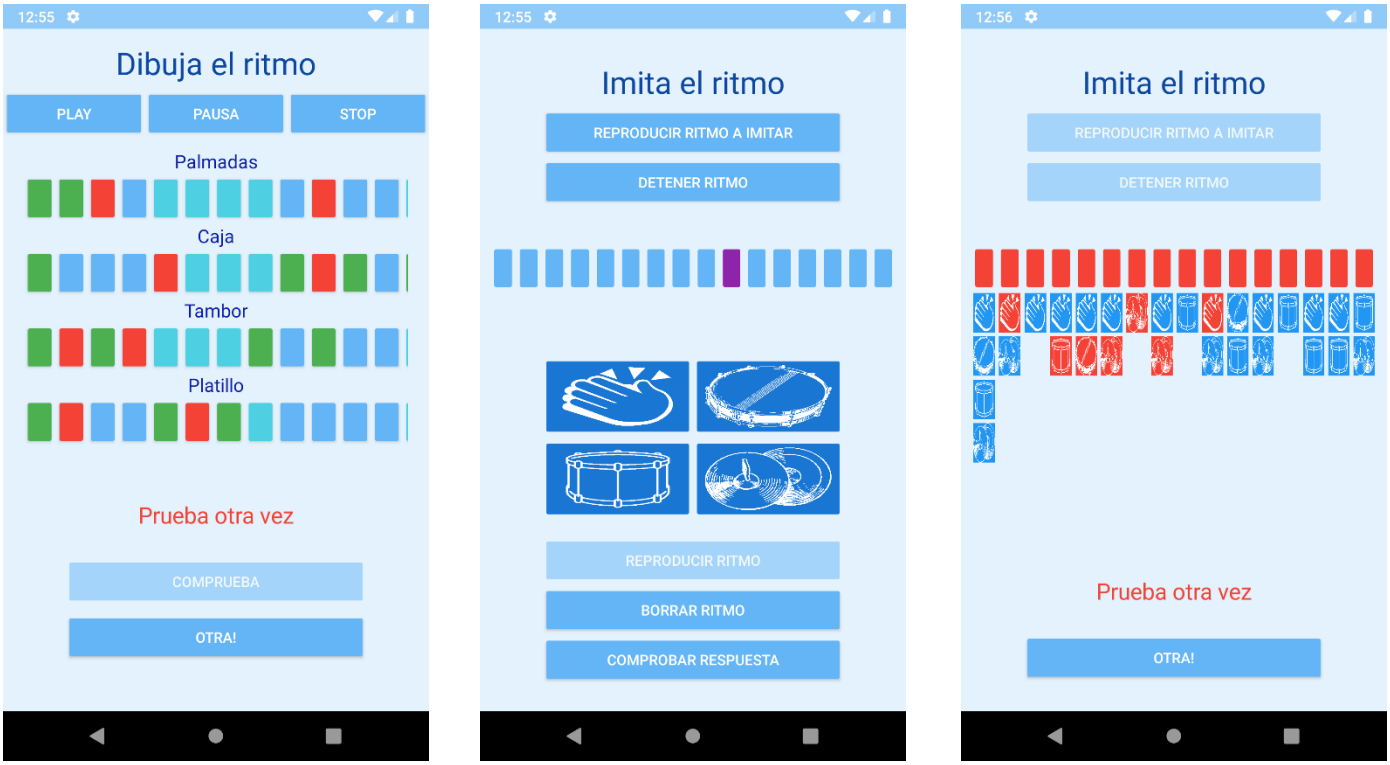
- Pantalla del modo "Dibuja Ritmos": justo debajo del título del modo, aparecen tres botones principales. "Play" reproduce el ritmo a dibujar en el nivel, "Pausa" detiene el ritmo que se está reproduciendo, pero lo deja justo en el momento dónde se encontraba y "Stop", que detiene el ritmo y lo reinicia.

  Más abajo hallaremos hasta cuatro filas de botones (depende del nivel en el que se esté jugando) junto a una pequeña leyenda para indicar el instrumento que representan. Los botones se encuentran dentro de un *ScrollView*, representando una línea temporal en la que cada uno actúa como un instante de tiempo dentro del ritmo. Por último, el botón "Comprueba" sirve para corregir la respuesta.

- Pantalla del modo "Imitar Ritmos": debajo del título del modo hay dos botones, uno para reproducir el ritmo del nivel que el usuario debe imitar y otro para detenerlo. A continuación, encontramos una fila compuesta de ocho o dieciséis botones (depende del nivel) cuya funcionalidad sigue siendo representar una línea de tiempo. Al reproducir un ritmo, el botón que representa el instante que suena se torna de color morado.

Más abajo hay cuatro botones que representan los instrumentos [25] a imitar (sólo se encuentran activos cuando el ritmo está sonando), y que el usuario debe pulsar en el instante que ese instrumento suena en el ritmo a imitar.

Por último, encontramos dos botones para reproducir y borrar el ritmo introducido por el usuario, y el botón para comprobar la respuesta. Al corregir, se muestran iconos de los instrumentos debajo de los botones que representan el ritmo, siendo el icono de color azul si el usuario no ha introducido el instrumento, verde si el usuario lo ha introducido correctamente y rojo si el usuario ha introducido un instrumento en un instante en el que no debía.



Pantallas de Ritmos

6.4.2.12 – Pantallas del modo mix

La interfaz del modo Mix se compone por una sucesión aleatoria de las pantallas de los otros modos de juegos a excepción del modo entonación con algunas variaciones.

- En lugar del botón para repetir el nivel, se muestra otro que permite avanzar a la siguiente pantalla.
- En la esquina inferior derecha se muestra al usuario el índice de la prueba.

De este modo, la interfaz está compuesta por un conjunto formado por dos pantallas de cada modo de juego, sumando un total de 14 pantallas de nivel.

Una vez que el usuario completa todas las pruebas, se muestra la pantalla de resultado del modo mix. En ella se muestran los resultados que ha tenido en cada tipo de prueba por separado, la suma total de aciertos, los aciertos requeridos para superar el modo y finalmente si el usuario ha conseguido superar o no el nivel.

Pantallas del Modo Mix

6.4.2.13 – Pantalla del modo notas

La pantalla del modo Notas presenta el título del modo, seguido de un conjunto de tres botones en los primeros niveles, dos en los niveles medios y solo uno en el nivel más difícil:

-Botón Nota: Reproduce el sonido a adivinar.

-Botón Referencia (LA) (Solo niveles 2-9): Reproduce un LA4 para ayudar al usuario.

-Botón Referencia (DO) (Solo niveles 2-3): Reproduce un DO4 para ayudar al usuario.

A continuación, se muestran las distintas posibles respuestas de la prueba, que aumentarán en número con el nivel. La opción seleccionada se oscurecerá.

En la parte inferior, la pantalla muestra el botón de Comprobar cuando se selecciona una opción, cuya funcionalidad es corregir la prueba resaltando en verde la opción correcta y en rojo la opción seleccionada si es incorrecta.

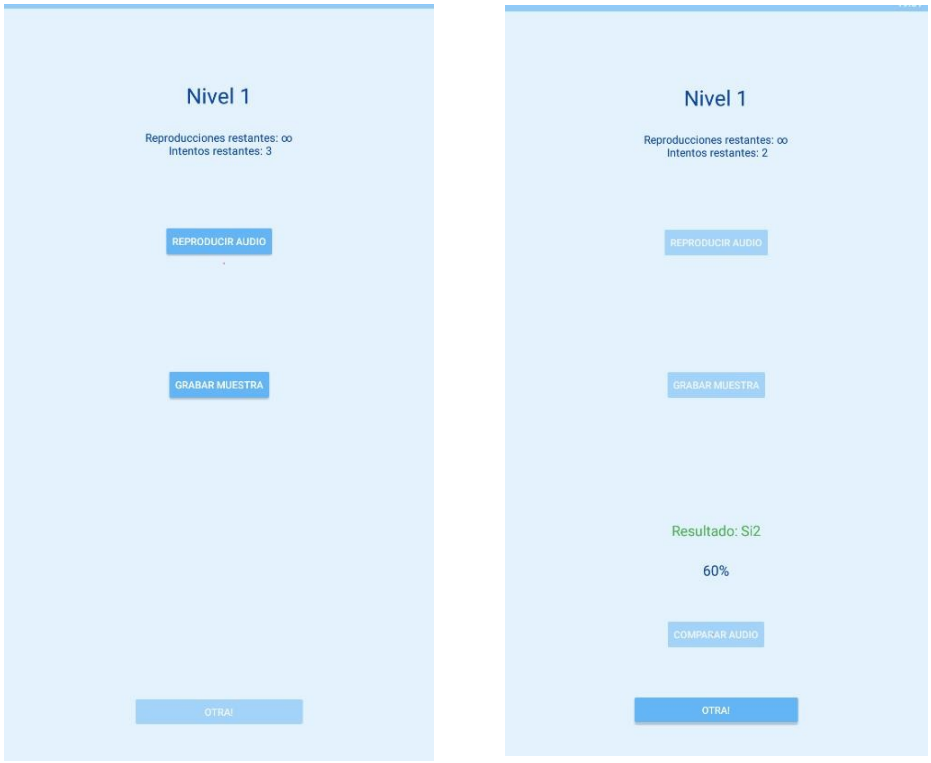
Pantallas de Adivinar Notas

6.4.2.14 – Pantalla del modo entonación

La pantalla del modo entonación consiste en un texto con el nivel en el que el usuario se encuentra, un botón para reproducir la nota a entonar y otro botón que activa un grabador durante cinco segundos en los que va recogiendo el sonido que realiza el usuario.

Una vez pasados esos cincos segundos, se muestra la nota que ha entonado el usuario, y un botón para comparar la nota que ha grabado el usuario con la nota que debía imitar. Una vez

comprobado, se muestra al usuario el resultado; si además ha acertado también se indica el porcentaje de afinación.



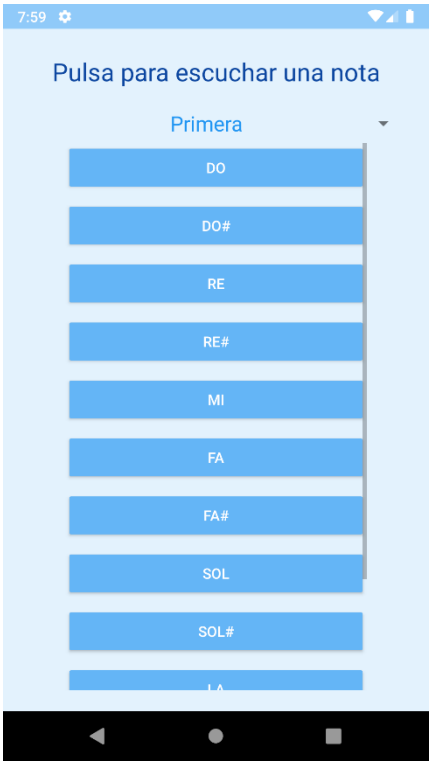
Pantallas de Entonación

6.4.2.15 – Pantallas de ayuda

Debido a que puede que no todos los usuarios tengan conocimientos previos respecto a música y sonidos de notas/acordes, se proporcionan distintas ayudas en los modos de juego que creímos conveniente. Estas pantallas se muestran tanto en la pantalla de selección de niveles como dentro del propio nivel, variando en función del modo de juego.

- Adivinar notas: se muestran en la pantalla de selección de nivel. El usuario puede escuchar en esta pantalla todas las notas de todas las octavas posibles.

- Intervalos: se muestran en la pantalla de selección de modo de intervalo. En ella el usuario puede escuchar todos los intervalos posibles construidos a partir de *LA4*.



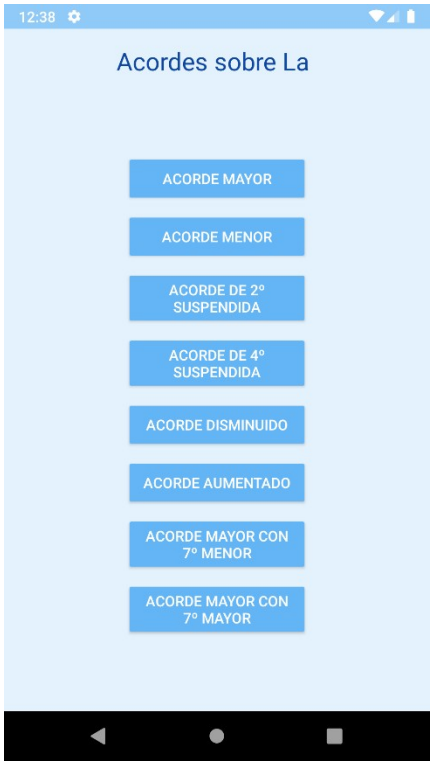
Pantallas de Ayuda de Notas

Pantalla de Ayuda de Intervalos

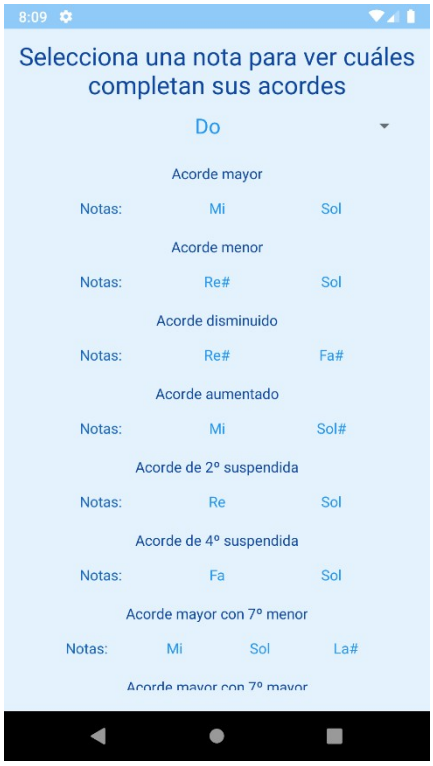
- Acordes: además de una pantalla de ayuda en cada pantalla de selección de nivel, comparten una pantalla adicional de ayuda dentro de la prueba en los primeros niveles. En esta pantalla adicional el usuario puede ver cómo suenan los acordes sobre la nota solicitada en la propia prueba. Las ayudas específicas de cada modo son:

    - Adivinar acordes: el usuario puede escuchar todos los acordes posibles construidos a partir de una nota inicial que él selecciona.

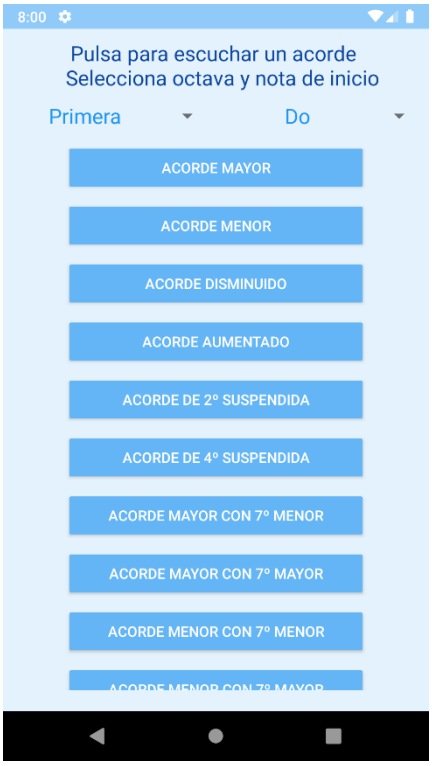
    - Crear acordes: el usuario puede ver qué notas completan cada acorde, eligiendo la nota de inicio de todos ellos.



Pantalla de Info de Acordes



Pantalla de Ayuda de Crear Acordes



Pantalla de Ayuda de Adivinar Acordes

6.4.2.16 – Pop ups de nuevo nivel y rango

Se trata de los pop ups que se muestran cuando el usuario sube o baja de nivel o de rango tras realizar una prueba, advirtiéndole del cambio de nivel o de rango. Los pop ups de nuevo nivel se dividen en dos tipos:

- Aquellos que se muestran cuando el usuario sube o baja nivel tras realizar una prueba.
- Aquellos que se muestran cuando el usuario ingresa por primera vez a un nivel. En este caso la funcionalidad principal es la de indicar los cambios que aparecen en este nuevo nivel con respecto al anterior.

Por otra parte, los pop ups de nuevo rango se muestran cuando el usuario desciende o sube de rango al comprobar una respuesta.

CAPÍTULO 7 – APORTACIÓN INDIVIDUAL

7.1 – ERNESTO PÉREZ MONTALVO

7.1.1 – Investigación con Kivy

Cuando tuvimos clara la aplicación que queríamos hacer y los primeros detalles al respecto, comenzamos a investigar los entornos y lenguajes que podíamos usar.

Debido a mi experiencia con Python en años anteriores comencé a buscar alguna biblioteca que nos pudiese ser útil para desarrollar una aplicación para dispositivos Android de forma intuitiva.

Encontré Kivy, una biblioteca de Python gratuita y de código abierto para desarrollar aplicaciones móviles y otro software de aplicaciones multitáctiles con una interfaz de usuario natural, y se intentó programar una aplicación de audio sencilla, un grabador/reproductor, que hiciese uso de la tecnología *PyAudio* y *NumPy*, dos bibliotecas que facilitarían mucho el trabajo en síntesis y análisis de audio.

Conseguí hacer funcionar la aplicación en el PC y el siguiente paso era generar el apk para probarla en el dispositivo móvil, para ello usé la herramienta *Buildozer*, pero antes, tuve que crear una partición de Ubuntu en mi portátil, ya que Buildozer no funcionaba óptimamente en Windows 10, cosa que más tarde me causaría severos problemas.

Una vez generada el apk lo instalé en el móvil para probarlo, pero no funcionó. Tras investigar y depurar ayudado por mis compañeros descubrimos que Buildozer presentaba una serie de incompatibilidades con Kivy, insalvables en la versión de Android de la mayoría de los dispositivos móviles.

Tras el fracaso en la investigación, continúe con la línea de investigación.

7.1.2 – Investigación de audio, TarsosDSP y notas

Cuando nos decidimos por Android Studio y Java para desarrollar la aplicación nos repartimos los trabajos, yo fui el responsable junto con Jesús de la investigación de las herramientas que podríamos usar para sintetizar y analizar audio del micrófono del dispositivo.

Tras investigar y desechar una serie de opciones como *MusicG [30]*, porque no funcionaban en Android Studio, encontré un GitHub con el TarsosDSP que proporcionaba procesamiento de audio en tiempo real en Java, a continuación, busqué alguna clase dentro del GitHub que nos pudiese servir, es decir, alguna clase capaz de extraer los hercios del tono escuchado por el micrófono, para de esta forma identificar la nota musical del tono y poder analizarla. Una vez encontrada se lo comuniqué a mis compañeros y, junto con Jesús, intentamos integrarla en nuestro proyecto de Android Studio, modificando lo necesario para que nos fuese útil en un dispositivo móvil, lo que requiso de investigación acerca del funcionamiento de Threads en dispositivos móviles.

Finalmente logramos hacer funcionar la clase para lo que nosotros queríamos, siendo capaz de captar sonido del micrófono y extraer los Hercios, a continuación, Jesús se encargaría de pasar dichos Hercios a notas musicales.

Una vez encontrada e integrada la herramienta para analizar el audio, encontré e integré en el proyecto una carpeta Drive pública que contenía ficheros .WAV de todas las notas musicales de la primera a la sexta octava, faltando la séptima, de la que más tarde se encargaría Ignacio. Dichos ficheros serían utilizados más tarde para reproducir las notas a adivinar en el modo de juego correspondiente.

7.1.3 – Implementación de Adivinar Notas, Crear Acordes y Crear Intervalos

Para la implementación de los modos de "Adivinar", Ignacio se encargó de Adivinar Intervalos y Adivinar Acordes y yo me encargué de Adivinar Notas, el más sencillo, Crear Intervalos y Crear Acordes. No obstante, ambos nos ayudamos mutuamente en la implementación de los métodos ya que en algunas ocasiones eran parecidos entre modos.

La implementación de Adivinar Notas fue sencilla, para ello use el método *FactoriaNotas*, diseñado entre Ignacio y yo. Debido a que la generación de una nota y las opciones posibles es trivial usando ese método no hubo ningún problema a la hora de implementar este modo.

La implementación de Crear Intervalos fue algo más complicada pues, además de usar *FactoriaNotas* para generar la primera nota del intervalo a crear, había que generar todos los intervalos posibles sobre esa nota, aplicar las restricciones correspondientes, escoger uno y generar las notas que completan ese intervalo, junto con todas notas que pueden completar cualquier intervalo posible sobre la nota de inicio. La última parte complicó la implementación de la clase, pues había que tener en cuenta muchas circunstancias distintas, lo que llevó a que surgiesen problemas con los casos extremos que no fueron solucionados hasta casi el final del desarrollo.

Finalmente, la implementación de Crear Acordes fue algo más trivial, ya que no había tantas circunstancias y casos extremos a tener en cuenta.

7.1.4 – Implementación del sistema de rangos y puntuaciones

Usando las propiedades de la base de datos creada por Ignacio, implementé un sistema de puntuaciones que permitiese al usuario subir niveles según aumentaba su puntuación. Para ello creé el DAO correspondiente para la base de datos, e implementé los métodos necesarios en el Gestor de la base de datos para que la puntuación del usuario en un determinado modo de juego aumentase o disminuyese tras fallo o acierto, desbloqueando así los nuevos niveles.

A continuación, ideé y establecí una serie de rangos de puntuación para cada modo de juego, que iban a corresponder con los rangos de nuestra aplicación, explicado anteriormente, y añadí las imágenes correspondientes a cada rango, para que se mostrasen al lado del botón del modo de juego pertinente.

7.1.5 – Implementación de los pop ups de nuevo nivel y rango.

En la fase final del desarrollo de la aplicación pensamos en añadir una serie de pop ups para hacer más intuitivo el uso de esta.

Para ello, y ante mi completo desconocimiento del uso de pop ups en Android, investigué la forma de implementarlos y algún diseño que nos pudiese ser útil para lo que queríamos hacer.

Una vez encontrada la forma, se lo comuniqué a Jesús para que hiciese los tutoriales de los modos e implementé los pop ups que avisaban al usuario de la subida o bajada de nivel y/o rango, así como los pop ups que informaban al usuario de los cambios de cada nivel nuevo con respecto al anterior y el pop up tutorial del sistema de rangos, escribiendo todo el texto correspondiente en el fichero Strings del proyecto.

7.2 – JESÚS VERDÚGUEZ GERVASO

7.2.1 – Idea

Al comienzo del desarrollo del trabajo de fin de grado se tenía claro que el tema principal iba a ser la música, ya que fue lo acordado en un principio con el director. Aunque nos ofreció distintas propuestas, siempre me ha gustado ser partidario de utilizar ideas propias para trabajos a los que se les va a dedicar semejante esfuerzo y tiempo. Por lo tanto, durante todo el verano de 2019 mi cometido principal fue encontrar un tema valido que juntase informática con música.

Es un hecho que las aplicaciones de aprendizaje musical siempre me han cautivado, ya que, como mencionamos al principio, muchas veces he estado interesado en aprender de forma autodidacta. Debido a esto, me enfoqué parcialmente en este campo, sin descartar otros por supuesto. Durante el verano comencé a preguntar a personas que se encuentran muy afiliadas a la música que, según su criterio y sin ningún tipo de condicionamiento, ¿qué tipo de aplicación necesitarían para mejorar su situación dentro de la música? Tras varios intentos, encontré la respuesta gracias a *Montse Egea*, que posteriormente participó en las entrevistas.

Montse, es chelista profesional y su respuesta fue: una herramienta para mejorar el oído musical. En base a que las mejores ideas vienen de una necesidad y mi, antes mencionado, gusto por las aplicaciones de aprendizaje autodidacta, decidí que este podría ser un buen tema para el trabajo de fin de grado.

Una vez puesto en común con el resto de miembros del grupo y con el director y estando todos conforme, contacté con diferentes perfiles dentro del mundo de la música, desde músicos profesionales y directores de orquesta, a músicos casuales y autodidactas, para así tener un gran abanico de opiniones y una buena base desde la que trabajar. Además, conseguí concretar algunas entrevistas para así poder hacer algunas preguntas más concretas e informarme en cuanto a teoría musical y como nos podría condicionar en el desarrollo.

7.2.2 – Primeros pasos en la implementación

Al comienzo de la implementación mi cometido fue buscar una buena manera de reproducir diferentes archivos de sonido a través de un dispositivo Android de forma que nos resultase sencillo cambiar de archivo y reproducir varios a la vez. Gracias a la página de desarrolladores de Android [56] encontré lo que posteriormente utilizamos, la clase *MediaPlayer*. Fueron realizadas las pruebas consecuentes para comprobar si realmente era lo que necesitamos y funcionaba de manera óptima.

7.2.3 – Modo Entonación

Con ayuda de la librería *TarsosDSP* se me asignó realizar el modo Entonación consiguiendo resultados fiables y siendo una actividad divertida para poder realizarla una y otra vez. En el momento en el que fue analizado en profundidad como la librería trataba el audio, utilicé los métodos que necesitábamos para realizar este modo de juego, concretamente el método *getPitch* que devolvía la frecuencia en hercios de los sonidos que captaba el micrófono del dispositivo. Con un estudio previo de cómo funcionan las notas musicales, qué es lo que hace que se diferencien unas de otras, etcétera. Comencé el desarrollo de este modo de juego.

Tuve ciertas dificultades, por ejemplo, al ser el primer modo de juego que implementé, Android funciona de una manera peculiar con los hilos de ejecución ya que el hilo principal es el encargado de la interfaz gráfica por lo tanto si se necesita interaccionar con la interfaz mientras se realizan otras actividades lógicas hay que tener especial cuidado. En estos casos entran temas como la cuenta atrás implementada en el modo Entonación, o el simple análisis de las frecuencias. Otro ejemplo sería calcular el porcentaje de afinación ya que la diferencia entre algunas notas y otras era realmente pequeño, por lo tanto, para mostrar al usuario cifras coherentes y que pudiese entender tuve que realizar un estudio matemático en base a los datos que teníamos en ese momento. [31]

7.2.4 – Modo Dibujar ritmos

Posteriormente desarrollé los dos modos de juego del paquete ritmos, primero implementé el modo *Dibujar ritmos*, y al ser el primero fue el más complicado de codificar. Ya que se introdujeron problemas como la sincronización de sonidos reproducidos por el dispositivo para que el ritmo tuviese cierto sentido o el cómo representar una línea de tiempo de un compás. [32]

Con ayuda del estudio de mercado que realizamos, pude encontrar solución a algunos de estos problemas, el resto fueron simples ideas que tuve y que por lógica debían funcionar y así lo hicieron.

7.2.5 – Modo Imitar ritmos

Posteriormente implementé el modo Imitar ritmos con ayuda de Ignacio, al haber implementado previamente el modo Dibujar Ritmos utilizamos bastantes cosas usadas previamente en ese modo que nos sirvieron también en este, aunque también hallamos ciertos problemas, el principal fue el cómo podríamos hacer que el usuario imite el ritmo que sonaba en tiempo real. Con

una combinación de hilos de ejecución e interfaz gráfica conseguimos codificarlo y con un resultado positivo ya que era un ejercicio dinámico y fiable.

7.2.6 – Pop ups de tutoriales y pantalla de información

Con ayuda de Ernesto y su explicación de como funcionaban los *Pop ups* en Android me dispuse a realizar los tutoriales de todos los modos de juego de *BasicMusic*. Mi objetivo era hacerlos de una manera que el usuario entendiese perfectamente cómo funcionaba el modo de juego, pero sin llegar a ser cargante y aburrido. Pensando en esto me dispuse a analizar los diversos tutoriales que he utilizado en otras aplicaciones e intente implementar los mejor de cada uno. Para ello realicé una copia de la interfaz de todos los modos de juego y mediante una ligera explicación y una serie de pantallas dinámicas, desarrollé la introducción a los modos de juego.

También en base a los conocimientos que había adquirido al realizar estos tutoriales realice el pop up de información de la pantalla principal.

7.2.7 – Diseño de imágenes de rangos

Queríamos que el usuario pudiera relacionar los rangos de *BasicMusic* con imágenes para que así fuese algo más visual. Yo personalmente con la aplicación de IPad *SketchBook [33]* diseñé las imágenes que consideramos ideales para este cometido.

7.3 – IGNACIO VÍTORES SANCHO

7.3.1 – Prototipo visual de la aplicación

En la fase de investigación, viendo que finalmente íbamos a realizar el proyecto en Android Studio, decidimos dividir la primera toma de contacto.

Mi trabajo en ese tiempo se centró en la realización de un prototipo visual en Java, para lo que me visualicé el curso del canal *Píldoras Informáticas [34]* en YouTube y los tutoriales de introducción de la API oficial de Android Studio.

Una vez que comprendí cómo se tenía que dividir el código en función de si consistía en apartado gráfico o lógica de la aplicación, les presenté a mis compañeros el prototipo y les hice un resumen básico de cómo funcionaban las llamadas entre funciones, así como la gestión de pantallas mediante los *Intent* y el *AndroidManifest.xml.*

Por último, para hacer los cambios de estilo de la manera más sencilla y rápida posible, investigué el uso de la carpeta *style* y busqué una paleta de colores sobre la que realizamos el resto del proyecto [35].

7.3.2 – Base de datos

Para el almacenamiento del progreso del usuario era necesario implementar, aunque fuese bastante simple, una base de datos que conservara los resultados de cada nivel. Buscando la manera más actual de realizar esto en una aplicación Android, encontré en la propia API de Android Studio que se recomendaba el uso de *Room. Room* proporciona una capa de abstracción para la gestión de la base de datos (internamente utiliza SQLite), de manera que la creación y gestión de las entidades se realiza mediante anotaciones, de una forma muy similar a *JPA*.

Una vez que comprendí cómo funcionaba, realicé un prototipo alternativo de una aplicación que gestionase una única entidad para comprobar que funcionaba correctamente, minimizando los posibles errores que heredasen de otros componentes de la aplicación.

Tuve un pequeño problema a la hora de editar la base de datos desde el mismo hilo de ejecución que gestionaba la interfaz de usuario, ya que por defecto esta opción no se permite porque puede afectar negativamente a la experiencia de uso ralentizando la aplicación. Sin embargo, como en nuestra aplicación la interfaz puede variar en función de los resultados, era imprescindible que se realizasen estos cambios de manera automática antes de seguir gestionando la vista (por ejemplo, si el usuario sube de rango, este cambio se tiene que mostrar en cuanto suceda). Esto se solucionaba modificando la función *create* de la clase *AppDatabase*, indicándole que se permiten cambios desde el hilo principal.

7.3.3 – Centralización de lógica empleando Singleton.

Como teníamos mucha funcionalidad que iba a ser común a la mayor parte de la aplicación, el patrón de diseño *Singleton* era perfecto para centralizar todo este código y evitar así la repetición del mismo. El esqueleto básico de estas clases (*FactoríaNotas, Controlador, GestorBBDD)* fue tarea mía, siendo ampliadas a medida que eran necesario por el resto de mis compañeros según los requisitos más específicos de cada modo.

7.3.4 – Pantallas de ayuda de los niveles y modos de juego.

Después de realizar todos los modos de juego y tener los requisitos ya cubiertos respecto a la funcionalidad de la aplicación, vimos que era necesario ofrecer distintas ayudas y guías al usuario para que pudiera ir avanzando sin ayudas externas a la aplicación.

Para ello, decidimos incluir distintas pantallas de ayuda en los modos de juego de la categoría *Adivinar* y en los primeros niveles de algunos de estos modos. Estas pantallas las hice con el objetivo de que el usuario pudiera reconocer los sonidos o las notas que se solicitan en las pruebas sin que fuese necesario ningún conocimiento musical previo.

7.3.5 – Implementación de Adivinar intervalos y Adivinar acordes.

Para la realización de estos modos, lo primero que tuve que hacer fue comprender los conceptos de intervalo y acorde, así como sus nombres y la diferencia entre ellos.

Primero realicé el modo de adivinar intervalos, para el que mantuve la temática de la prueba *Adivinar Nota* adaptando el botón de reproducir nota. Para la reproducción del intervalo con un único *Media Player,* hice varias pruebas hasta dar con la duración idónea del silencio entre las notas.

Para el modo *Adivinar Acordes* seguí el mismo modo de trabajo que con el modo *Adivinar Intervalos*, pero esta vez apareció el problema de conseguir coordinar las notas ya que la reproducción se realizaba con distintos *Media Player*. Para arreglarlo, debido a que el procesamiento de los ficheros de audio no era inmediato, decidí que lo mejor era realizar el procesamiento de todos los reproductores para, a continuación, llamar al método *play* por medio de un bucle.

7.3.6 – Solución problema gestión y bloqueos de MediaPlayer.

Una vez que teníamos la aplicación en una fase avanzada y comenzamos a compartirla a gente para que nos dijeran posibles mejoras o fallos, uno de estos fallos consistía en que la aplicación se bloqueaba cuando se pulsaban los botones que reproducían audio muchas veces. Esto se debía a que no gestionábamos de manera correcta los MediaPlayer, quedando una pila de MediaPlayer en la memoria y provocando que esta se saturase. Para solucionarlo, cree la clase *Reproductor* en la que se centralizó la lógica de reproducción de audio e investigué cómo se podía liberar la memoria empleada para los MediaPlayer una vez que estos finalizaran su reproducción.

7.3.7 – Implementación del Modo Mix.

Este fue el último modo que se implementó en nuestro proyecto, debido a que nuestro director nos comentó que le parecía la funcionalidad más interesante.

Para gestionar el orden y la aparición de las distintas pruebas, vi que lo más conveniente era realizar un controlador exclusivo que se apoyase en el controlador general para conseguir la dificultad individual de cada prueba. Este nuevo controlador sería el encargado de almacenar el resultado de cada prueba y el índice de la prueba que se estuviera realizando en cada momento, así como el resultado final del modo. Para dividir la dificultad de los niveles, cree un enumerado que almacenase el nivel de cada prueba que formaba el modo y el número de aciertos necesarios para considerar el Mix superado.

Debido a cómo está pensado el ciclo de vida de una *Activity*, había problemas para lanzar las distintas pantallas una tras otra, sin que se solaparan. Gracias a mi compañero Ernesto, descubrimos un modo alternativo de comenzar las actividades que permitía dejar en espera la actividad que coordina estas llamadas.

CAPÍTULO 8 – CONCLUSIÓN

8.1 – DISPONIBILIDAD DEL PROTOTIPO E INSTALACIÓN

El prototipo de *BasicMusic* se puede encontrar y descargar en *GitHub,* en el enlace https://github.com/ivs95/BasicMusic.

En cuanto a la instalación, hay varias maneras de realizarla:

Si se usa el IDE *Android Studio* con el proyecto descargado de *GitHub*, se puede depurar el proyecto y por lo tanto hacer una instalación temporal del proyecto, ya sea en un dispositivo *Android* (activando la opción de depuración mediante USB), en un emulador externo (como *BlueStacks [36]*) o en el emulador local que ofrece *Android Studio*.

También con *Android Studio* se puede realizar la creación del archivo APK a partir del proyecto y de esa manera instalarla tanto en un dispositivo como en un emulador.

Para facilitar la prueba de todos los niveles de los modos de juego hemos creado un usuario de prueba que tiene todos los niveles desbloqueados. Las credenciales son:

Usuario: usuario@prueba.com

Contraseña: 1234

8.2 – TRABAJO FUTURO

Una de las cosas de las que más orgullosos estamos, es la gran escalabilidad que tiene *BasicMusic*, ya que la música como ámbito de aprendizaje es inmenso. Por lo tanto, nuestra aplicación podría mejorar prácticamente de manera infinita.

Hay varias funcionalidades que tuvimos en cuenta a la hora de desarrollar el proyecto y que por varias razones no llegamos a implementar completamente. En concreto:

- Melodías: este es un punto que podría ayudar a mejorar notablemente *BasicMusic,* que consiste en adaptar los modos *Adivinar Notas* y Entonación para melodías, ya que en general un músico trabaja con melodías.

  Una forma sería que la aplicación reproduciría una melodía con un determinado número de notas y el usuario tuviese que adivinar cuáles son. En el modo *Entonación* la aplicación podría ofrecer al usuario la posibilidad de que en un ejercicio tuviese que entonar una melodía de varias notas.

- Temas musicales conocidos: basándonos en el anterior punto de melodías, creemos que un gran incentivo para el usuario sería poder adivinar o imitar las notas de sus

canciones favoritas. Así ellos podrían interpretarlos ya sea mediante un instrumento o su propia voz.

- Modos de juegos: creemos que hay prácticamente infinitas maneras de aprender, sobre todo en un campo tan versátil como es la música. Por lo tanto, creemos que se pueden crear multitud de modos de juego diferentes a los ya implementados, ya sea tratando los temas que hemos contemplado u otros completamente diferentes, por ejemplo, la interpretación de un instrumento, el solfeo u otros muchos ámbitos dentro de la música.

8.3 – CONCLUSIONES FINALES

Una vez concluido el proyecto, nos hacemos la siguiente pregunta: ¿hemos conseguido el objetivo que nos propusimos?

Nuestra respuesta es simple: rotundamente sí. Hemos conseguido crear una aplicación para dispositivos Android divertida y útil, tanto para usuarios sin ningún tipo de conocimiento musical como para usuarios experimentados en el campo que necesitan perfeccionar su oído musical o su entonación de la voz.

Además, no solo hemos conseguido los objetivos que nos planteamos en un principio, sino que hemos ido mucho más allá.

En cuanto al oído musical, no nos hemos centrado en un solo modo de juego, en su lugar, hemos creado un conjunto de siete modos de juego para darle al usuario bastante variedad y, al mismo tiempo, darle la posibilidad de practicar diversos elementos musicales como los intervalos, acordes y ritmos.

En cuanto a ritmos, en ningún momento al comienzo del proyecto nos planteamos ejercicios para mejorar el sentido del ritmo de nuestros usuarios, y con los dos modos de juego enfocados en ritmos creemos que puede llegar a conseguirse.

Por otra parte, centrándonos en la entonación de la voz, hemos creado un modo de juego bastante fiable en diversos entornos que no solo te dice si efectivamente has entonado la nota que te pedimos, sino que además te dice el porcentaje de afinación de esa nota.

Otro de los objetivos era que la aplicación fuese divertida, para que así el proceso de aprendizaje fuese llevadero; creemos que lo hemos conseguido mediante la implementación del sistema de niveles y rangos. El sistema de niveles, además de hacer la dificultad de la aplicación escalable y viable para todo tipo de usuarios, genera expectación en el usuario por saber qué se va a encontrar en el siguiente nivel. Los rangos le dan a la aplicación un grado de cierta competitividad, ya sea con uno mismo o con diferentes amigos que también usen la aplicación.

Por lo tanto, como hemos mencionado al principio, *BasicMusic* es una aplicación Android, de aprendizaje musical, útil para todo tipo de usuarios y divertida.

CHAPTER 9 – IN CONCLUSION

9.1 – PROTOTYPE'S AVAILABILITY AND INSTALLATION

The BasicMusic prototype can be found and downloaded on GitHub, in the link https://github.com/ivs95/BasicMusic.

We chose GitHub as the version control repository because of its ease of use, features, and backwards compatibility with the IDE we used in development.

About the installation, there are several ways to do it:

If the Android Studio IDE is synchronized with the project from GitHub, you can debug the project and therefore do a temporary installation of the project, either on an Android device (activating the debugging option via USB), in an external emulator (such as BlueStacks) or in a local emulator offered by Android Studio.

Furthermore, with Android Studio you can create the APK file from the project and thus install it on both a device and an emulator.

To facilitate testing at all levels of the game modes, we have created a test user that has all levels unlocked. The credentials are:

User: usuario@prueba.com

Password: 1234

9.2 – FUTURE WORK

One of the things that we are most proud of is the great scalability that BasicMusic has, since music as a learning environment is immense. Therefore, our application could improve almost infinitely.

There were several things that we considered when developing the project and we did not fully implement for several reasons. Specifically:

- Melodies: Melodies could help to improve BasicMusic considerably. We could made it work by adapting the Guess Notes and Intonation modes, so that they use melodies instead of individual notes, since musicians usually work with melodies. We think that it would be a great improvement if the application could offer the option of, for example, playing the polyphonic melody so that the user had to guess which are the notes of the melody. In the Intonation mode, the app could offer the possibility of singing melodies composed of several notes.

- Known songs: based on the previous point, we think that a great incentive for the user would be to be able to guess or imitate the notes of their favorite songs. Thus, they could interpret them either through an instrument or their own voice.

- Game modes: we believe that there are practically endless ways to learn, especially in a field as versatile as music. Therefore, we believe that a lot of new game modes can be created, for example, the interpretation of an instrument, solfege or many other areas within music.

9.3 – FINAL CONCLUSIONS

Once the project was done, we wonder the following: have we achieved the objective that we set ourselves?

Our answer is simple: flatly yes. We have managed to create a fun and useful application for Android devices, both for users with no musical knowledge whatsoever and for users experienced in the field who need to improve their musical ear or voice intonation.

In addition, we have not only achieved the objectives that we set ourselves at the beginning, but we have gone much further.

When it comes to the musical ear, we have not focused on a single game mode. Instead, we have created a set of seven game modes to give the user enough variety and, at the same time, give him the possibility of practicing various musical elements like intervals, chords and rhythms.

When it comes to rhythm, at no time at the beginning of the project we considered exercises to improve the sense of rhythm of our users. However, we ended up with two modes of play focused on rhythms that we believe can be very helpful.

As for the intonation of the voice, we have created a reliable game mode in various environments that not only tells you if you have correctly matched the given note, but also tells you the pitch percentage of that note.

Another objective was to make the application fun, so that the learning process was easy; We believe that we have achieved this by implementing a progression system of levels and ranges. The level system, in addition to making the difficulty of the application scalable and viable for all types of users, generates expectation in the user to know what will be found in the next level. The system of ranges gives the application a certain degree of competitiveness, either with oneself or with different friends who also use the application.

Therefore, as we mentioned at the beginning, BasicMusic is an Android application, for music learning, useful for all kinds of users and fun.

CAPÍTULO 10 – BIBLIOGRAFÍA

[1] FL STUDIO. (2020) URL:

HTTPS://WWW.IMAGE-LINE.COM/FLSTUDIO/

[2] BANDLAB. (2020) URL:

HTTPS://WWW.BANDLAB.COM/

[3] SMARTICK. (2020) URL:

HTTPS://WWW.SMARTICK.ES/

[4] MUSIC TUTOR. (2018) URL:

HTTPS://WWW.MUSICTUTORAPP.COM/

[5] MUSIC TRAINER. (2013) URL:

HTTPS://PLAY.GOOGLE.COM/STORE/APPS/DETAILS?ID=COM.PROFITTLICH.ANDROID.MUSICTRAIN&HL=ES_419

[6] COMPLETE RHYTHM TRAINER. (2020) URL:

HTTPS://WWW.COMPLETERHYTHMTRAINER.COM/

[7] COMPLETE EAR TRAINER. (2020) URL:

HTTPS://COMPLETEEARTRAINER.COM/ES/

[8] MUSIC TRAINER PROFESSIONAL. (2020) URL:

HTTPS://PLAY.GOOGLE.COM/STORE/APPS/DETAILS?ID=COM.MARTINS.MARTIN.MUSICTRAINERPROFESSIONAL2&HL=ES

[9] MONTSERRAT EGEA TAPETADO. (2019) URL:

HTTPS://WWW.ESCUELASUPERIORDEMUSICAREINASOFIA.ES/ALUMNO/MONTSERRAT-EGEA-TAPETADO

[10] ORQUESTA SINFÓNICA DE RTVE. (2020) URL:

HTTPS://WWW.RTVE.ES/ORQUESTA-CORO/ORQUESTA-SINFONICA/

[11] ORQUESTA SINFÓNICA MUSIKENE. (2020) URL:

HTTPS://MUSIKENE.EUS/

[12] ESCUELA SUPERIOR DE MÚSICA REINA SOFÍA. (2019) URL:

https://www.escuelasuperiordemusicareinasofia.es/

[13] Apuntes de la asignatura de Informática Musical de la Facultad de Informática de la Universidad Complutense de Madrid, Profesor Miguel Gómez-Zamalloa Gil. (2020).

[14] Teoría Musical. (2020) URL:

https://es.wikipedia.org/wiki/Comp%C3%A1s_(m%C3%BAsica)

https://www.ladomicilio.com/es/cursos-de-musica/teoria-musical/notas-musicales

https://es.wikipedia.org/wiki/Acorde#Acordes_mayores

[15] Pyaudio. (2006) URL:

https://people.csail.mit.edu/hubert/pyaudio/docs/

[16] Numpy. (2020) URL:

https://numpy.org/

[17] Kivy. (2020) URL:

https://kivy.org/#home

[18] Github - Buildozer. (2020) URL:

https://github.com/kivy/buildozer

[19] JUCE. (2020) URL:

https://juce.com/

[20] Android Studio. (2020) URL:

https://developer.android.com/studio

[21] JFugue. (2017) URL:

http://www.jfugue.org/

[22] Github - TarsosDSP. (2019) URL:

https://github.com/JorenSix/TarsosDSP

[23] Android Studio - Room. (2020) URL:

https://developer.android.com/training/data-storage/room

[24] Creación dinámica de botones. URL:

http://umhandroid.momrach.es/creacion-dinamica-de-botones/

[25] Imágenes botones ritmos. URL:

https://www.goodfreephotos.com/vector-images/snare-drum-vector-clipart.png.php

https://freesvg.org/surdo-outline

https://commons.wikimedia.org/wiki/File:Emojione_BW_1F44F.svg

https://pixabay.com/es/illustrations/placas-de-choque-instrumento-musical-3376166/

[26] Android Studio - Thread. (2019) URL:

https://developer.android.com/reference/java/lang/Thread

https://www.guru99.com/multithreading-java.html

[27] Android Studio - AudioFX. (2020) URL:

https://developer.android.com/reference/android/media/audiofx/AcousticEchoCanceler.html

https://developer.android.com/reference/android/media/audiofx/NoiseSuppressor

https://developer.android.com/reference/android/media/audiofx/AudioEffect.html

[28] Android Studio - Media Player. (2020) URL:

https://developer.android.com/reference/android/media/MediaPlayer

[29] Repositorios de archivos WAV. URL:

http://freepats.zenvoid.org/

[30] Github - MusicG. (2017)

https://github.com/loisaidasam/musicg

[31] Java2s - Timer. URL:

http://www.java2s.com/Tutorials/Java/Java_Utilities/How_to_use_Java_Timer_and_TimerTask_to_schedule_a_task.htm

[32] StackOverflow - Reproducción de varios archivos en orden. (2016) URL:

https://stackoverflow.com/questions/10916129/playing-multiple-files-in-mediaplayer-one-after-other-in-android

[33] Sketchbook. (2018) URL:

https://sketchbook.com/

[34] Curso de Android Studio. Píldoras Informáticas. (2019) URL:

https://www.youtube.com/watch?v=pdYkmCcQFd8&list=PLU8oAlHdN5Bkn-KS1sRFlSEnXXcAtAJ9P

[35] Github - Material Design. (2015) URL:

https://github.com/JavierSegoviaCordoba/Resources/blob/master/XML/material_design_colors.xml

[36] BlueStacks. (2020) URL:

https://www.bluestacks.com/es/index.html